Aspectually Augmented Models

Tom Wallis

Submitted in fulfilment of the requirements for the Degree of Doctor of Philosophy

School of Computing Science

College of Science and Engineering

University of Glasgow

Table of Contents

Todo list

Chapter 1

Introduction

1.1 The Problem to Solve

Consider a research software engineer, responsible for the development and maintenance of a well-adopted model of some socio-technical system — perhaps a public transport system, the spread of a disease in a pandemic, or a software development team's day-to-day work. As the model's use spreads and users' requirements broaden, how might our engineer manage their maintenance burden? What tools and techniques might they make use of to ease their task? Their tool might have originally modelled the interactions of individual developers in a team (to extend the latter example), and offered parameters for development methodologies or team sizes; subsequent researchers might look to run simulations of the team with biased behaviour such as illness or tiredness, or different experience levels in the team, or communication breakdowns between individuals, or the impact of a change in management or project direction. What then?

Typically, if a required change would alter the model significantly, researchers looking to adopt the tool for new purposes would fork it (assuming the source is available to them under a permissive enough licence) and would make whichever modifications suited their needs. Other researchers might make their own forks. If the change is small, or if the new behaviour can be enabled in configuration, then it might be merged into the original tool. The contribution is thus disseminated to the research community who adopted the original tool without the need to migrate to a fork. However, if a future team wanted to research a combination of factors — say, simulating the impact of a change of management on teams with different levels of experience and communication quality — they would repeat the process, producing another fork with their own implementation of this particular combination. As modifications are built on top of each other, the logic for each possible behaviour is interwoven with that of the original model.

Such a codebase would be increasingly difficult to maintain. For example, the abstractions used to modularise it may have elegantly separated different concerns into different modules at first, but abstractions are often domain-specific; as the model becomes increasingly general-purpose, the abstractions used to separate concerns accumulate technical debt in the face of broader use-cases. In addition, the behaviours added by different teams may not make sense when enabled at the same time. In the example of a model of a software development team, a change representing remote work and a change representing the spread of a virus in different office settings are at odds conceptually and both are unlikely to be required in most cases. As a result, the tool's source code risks becoming confusing and unwieldy.

There is therefore an opportunity to improve the design of models and their codebases. We identify a need for a technique which allows a model to be altered in any arbitrary fashion so that the changes somebody could make to the tool can be represented as separate, optional modules — even when that change introduces or alters logic in the middle of an existing process. It should also permit a developer to select the changes they need for their study, effectively composing a model with whichever behaviours they require.

1.2 A Proposed Solution

We identify aspect-oriented programming as a tool which is well-suited to the needs of research software engineers who build models as described earlier. This section gives a high-level introduction to aspect-oriented programming concepts. A more detailed discussion of the aspect-oriented paradigm and its literature follows in chapter 2.

The systems being modelled throughout this thesis are socio-technical systems, which involve the interaction between human actors and technology & processes. While we expect this technique is broadly applicable to models & simulations generally, the scope of the research is reduced to examine it as applied to socio-technical models in particular; these are also briefly introduced.

Having discussed aspect-oriented programming and the modelling & simulation of socio-technical systems, we section concludes by examining the technique's possible application in the context of the problem described.

1.2.1 What is Aspect-Oriented Programming?

Aspect-oriented programming originated in Kiczales et al.'s work in Xerox Parc [Kic+97]. The motivation for the paradigm was the modularisation of "cross-cutting concerns": parts of a program which are not directly related to the fulfilment of its requirements but are common throughout it. Cross-cutting concerns generally doesn't relate to their neighbouring logic semantically. Rather, they fulfil an ancillary task, such as logging. This suggests it is a suitable candidate for modularisation: the single responsibility principle [Mar03] suggests that software with multiple concerns should be refactored into separate modules. However, traditional techniques are unsuitable for the creation of separate modules in these situations.

Aspect-oriented programming introduces a new pattern which modularises cross-cutting concerns. This is achieved by separating them from a program's core logic and rewriting them as modules — termed "advice", the combination of some implementation (an "aspect") and a place to apply it (a "join point") — which are later "woven" back into their intended positions by ensuring the aspect's logic is included somehow at its corresponding join point. This allows high-level behaviours such as logging, synchronisation, or memory safety to be factored away from a program's logic and into separate units. Unlike traditional modules, which are included by the code which requires them, a weaver composes advice and the code they should be applied to, with neither requiring knowledge of the other. This separation is core to aspect-oriented programming's design, and is referred to as "obliviousness".

Maybe introduce pointcuts here too — although I don't think it would add much to the discussion

Aspect-orientation allows for existing codebases to be structured in new ways, but its notion of modularising cross-cutting concerns is general and can be applied in other fields too. For example, business process modelling languages have been adapted to accommodate behaviours or processes modelled using advice [Cap+09; Sil+20; CM07]. Some researchers suggest using aspect-oriented programming to separate concepts in experimental software, such as experiment and observation, to mirror the setup of a traditional experiment in research-specific codebases [GK99]. Concepts originating in aspect-oriented programming have also seen use outside of software engineering [Cie+11; Cap+09; Sil+20; CM07].

This can be reworked to refer to variance less in line with the approach to discussing the research I now take in chapter 6 and chapter 7. However it's not essential, so I'll revisit *if time permits.*

This thesis explores modelling changes to behaviour using advice, and is applied to the modelling of socio-technical systems in particular. Socio-technical systems partly consist of the behaviour of human agents within a system, which can exhibit variations contingent on environmental factors and other states. This variance might be expressed differently in different agents in the system. In later chapters, we demonstrate that these behavioural variations can be separated from a generic model into advice, with the advantage that the generic model's codebase is simplified and any particular behaviour of interest in a particular study can be modelled by weaving those behaviours at runtime. Advice, as demonstrated in later chapters, is a flexible and useful tool for composing models.

1.2.2 Challenges in Aspect-Orientation Today

The aspect-oriented paradigm has some drawbacks. Identifying these means they can be addressed when applying aspect-oriented programming to a new domain where possible, and generally contextualise the research in later chapters. While they are discussed in detail in section 2.1.7, a quick summary is given here as background for this chapter's later explanation of this thesis' aims and contributions (see **??** and section 1.3).

The paradigm's philosophy that advice and the code it is applied to are oblivious to one another produces codebases which can be tricky to comprehend. When reading the implementation of some function or feature, it's not possible to know whether advice is being applied to it elsewhere. If it is, the behaviour of that code might be altered. Accurately understanding a program's control flow can therefore be challenging as a consequence of aspect-oriented programming's fundamental design choices.

Other limitations of aspect-oriented programming concern advice itself. Aspect orientation tooling typically allows for aspects to be woven before their join points, after them, or around them (in effect, both before and after). However, this requires aspects to treat their join points as black boxes: while introspection might reveal some details about them, logic cannot be added within them. Many codebases are not designed with the intention of applying aspects, meaning that there may be no suitable join points to alter behaviour in the way a developer intends.

Lastly, while a variety of toolsets exist for aspect-oriented programming, few studies appear to have been undertaken confirming that its expected benefits actually materialise. Some scepticism around its practical benefits have been raised [Ste06; Prz10; CSS04]. An application of aspect orientation to a new domain ought to demonstrate that the technique is of practical benefit, rather than a theoretical curiosity.

1.2.3 What are Socio-Technical Systems?

The research in this thesis applies aspect-oriented programming to the modelling & simulation of socio-technical systems. A socio-technical system is one constituted of people, technology, and the interactions between the two. The term originates in the study of work and organisations undertaken by Trist and Bamforth [TB51]. Socio-technical systems research continues to focus on organisations and workplaces [Pas+19; BS11], though the interaction of people and technology is also studied more broadly in areas such as degraded modes [JS07], resilience engineering [HWL06], and responsibility modelling [LSS09].

The involvement of human behaviour in the system's dynamics makes these systems a useful subject for aspect-oriented modelling, as variations in the behaviour of an individual or an entire group is a candidate for cross-cutting concern. To this end, models of learning as cross-cutting concerns are developed in chapter 6. In addition, datasets of users' interactions with computer systems can be collected to empirically verify models of their behaviour. This is also useful for the research in later chapters: chapter 5 describes the design, implementation, and data collected from the release of a mobile game, another example of a socio-technical system.

The technique of creating models using aspect-orientation to modularise particular behaviours or dynamics is not exclusively socio-technical in nature. We suspect that the technique can be generalised to the behaviours and dynamics of an arbitrary system in an arbitrary field; however, the paradigm is demonstrated as a socio-technical modelling technique for the purposes of setting bounds on our aims. It is not feasible to verify the technique's appropriateness in *every* type of system, but it is feasible to investigate the technique as applied to a *particular* type of system. Also, aspect-orientation has received attention in the business process modelling research community [CM07; Cap+09; CM06], setting a precedent for its broader use in modelling other socio-technical systems too.

1.2.4 Simulations & Models

The field of simulation & modelling concerns the building of models of some system and the simulation of that system using the model. We have found (anecdotally) that the difference between the two can be vague; some definitions are provided to avoid confusion.

In this thesis,"model" will be used to refer to some representation of a system or subject of study, or an abstraction of it. In this sense, a model can be a concept, a physical model, a diagram, and so on — in this thesis in particular, "model" will be used with the meaning: *"a software representation of a system or other subject of study"*. The term "simulation" also requires definition, and throughout this thesis will generally refer to the execution of a model. More formally put, "simulation" is used to refer to *"the emulation of the processes within a system or other subject of study"*. A simulation typically generates data, such as a record of system states over time or a log of actions taken by some actor within that system.

As the research in later chapters is primarily concerned with the simulation & modelling of socio-technical systems, the subject of a simulation or model should be assumed to be a socio-technical system unless otherwise specified.

1.2.5 Possible Benefits of Aspect-Oriented Modelling

Some specific benefits we anticipate from an aspect-oriented approach to building and maintaining models include:

Advice as Units of Model Change Scientific models which already exist are difficult to modify. They can be adopted by many research groups, meaning breaking changes impact the broader community; they can be the basis of published results, so changing the source code might invalidate the relationship between ongoing work and published work; and they can be brittle, as the incentive when writing software for research purposes is to achieve results worth publishing, rather than to produce a high-quality and maintainable codebase. This differs from a commercial software engineering team's incentives, which are typically to produce software which they can continue to produce in the future with minimal overhead imposed by code quality. For these reasons, scientific codebases have special requirements which discourage direct modification, particularly for different use-cases. Aspect-oriented programming allows updates to codebases to be written without direct modification to the source code. In the case of research software specifically, this has been shown to achieve positive results in previous case studies [Ion+09].

Advice as Tools for Instrumenting Scientific Codebases Models of a system are ideally concerned with the logic required to accurately model the system itself, and do not contain additional logic to instrument the model for the purposes of a particular experiment. This is desirable because it allows the model to be re-used for many experiments, as the instrumentation to make observations for a particular purpose are not woven throughout its logic. This also makes the codebase easier to read: a researcher interested in it must only read the logic required to implement a model, and doesn't also have to identify the parts of the program which don't model a system at all, but produce observations from a simulation based on the model. Finally, the separation of observational apparatus from a model mirrors the design of a traditional experiment, where observations are carefully made so as not to bias results. This potential use-case of aspect orientation has been suggested in the community [GK99].

Advice as Hypothesised System Behaviour If advice is woven into a model, side effects of the advice could be used to alter the model. In this scenario, advice could be written which changes a model to have a desired effect, such as correcting an error in the model or updating it in light of new research. Moreover, an established model may serve as a "proving ground" for future research: hypothesised behaviours in a system may be written as updates to a proven model of that system, and its predictive quality compared to the model executed without advice woven. A simple experimental structure is yielded by this process. Given advice may be useful as a unit of model change, these units may be proposed, tested, and combined to produce a variety of models to meet different research groups' needs without requiring onerous maintenance efforts.

1.3 Contributions

1.3.1 Research Questions

The research questions investigated in the following chapters are:

> **RQ1** *Is an aspect orientation tool which is appropriate for use in simulation & modelling feasible to design?*
>
> **RQ2** *Can models of systems more accurately reflect their subjects through advice-based improvements?*
>
> **RQ3** *Can advice be used to accurately introduce behaviours or parameters into a model which were not originally present in it?*
>
> **RQ4** *Can advice be used as a portable module, such that aspect-oriented improvements to one model can be woven into another without loss of performance?*

These are motivated following a review of relevant literature in section 2.4.1.

1.3.2 Summary of Contributions

In developing tooling for aspect-oriented modelling and investigating the technique's feasibility, this thesis makes multiple contributions to the relevant research communities, including but not limited to answering the above research questions. These are summarised as:

Tooling for Aspect-Oriented Modelling This thesis presents a redesigned and re-implemented version of a tool, `PyDySoFu`, originally prototyped for some prior work,[1] which contributes both a tool which can be employed for aspect-oriented modelling and a weaving technique designed for legibility when applying aspects to models. It can be applied to existing Python code with no modification to target code, and introduces minimal dependencies when added to a project.

[1] See chapter 3 for a discussion on related research undertaken prior to starting this PhD.

Dataset describing Socio-Technical System Interaction A dataset describing 370 players' interactions with a mobile game released for research purposes called RPGLite. The design, implementation, and data obtained from the game is presented, and the dataset collected via RPGLite is used in experimental design for this thesis.

Demonstration of Aspect-Oriented Model Enhancement Aspects are used to augment a model of RPGLite gameplay and demonstrate that aspect-oriented enhancements can alter a model to improve it (according to some metric). In particular, we show that a model can be augmented to synthesise data with the properties of the real-world dataset.

Using Aspect-Oriented Enhancements to Identify Hypothesised Player Behaviour A model of learning is presented with multiple parameters. We demonstrate that this can be tuned to specific real-world players' behaviour using the dataset collected, resulting in player-specific models of learning. Properties of learning of specific real-world players are identified by fitting a learning model to their RPGLite gameplay data.

Investigation into Aspect Portability We investigate whether aspect-oriented changes to a model can be ported from one system to another, taking advantage of aspect-orientation's modular nature.

Exploration of Opportunities Enabled by Aspect-Oriented Modelling New research opportunities are yielded by this novel technique in simulation & modelling. As in other PhD theses where contributions carry potential for many pieces of future work [Mar94a], the avenues for future research are broad enough that identifying and enumerating them is a substantial piece of work and constitutes an additional contribution.

1.4 Thesis Structure

The rest of the thesis is structured as follows.

chapter 2 surveys the project's relevant literature and identifies specific research questions in the field which the thesis addresses. Some earlier work precedes the research presented in this thesis; to delineate earlier contributions from those presented in later chapters, chapter 3 surveys the state of the research project before this PhD began.

Having surveyed the literature, identified research questions, and established the starting point of the research, chapter 4 describes the re-design and implementation of an aspect-oriented modelling tool, PyDySoFu. Other technical contributions follow in chapter 5, which describes the design and implementation of RPGLite, a mobile game developed for research purposes, as well as the data collected from it.

Later chapters explore the application of aspect-orientation of aspect-oriented programming to simulation & modelling codebases. Three experiments are constructed using a model of RPGLite to answer the latter three research questions. The experiments involved share the majority of their technical foundations and methodology, but are used in different ways to research different facets of aspect-oriented programming as applied to simulation & modelling. For this reason, their common foundations and methodology are explained in chapter 6. The first research question is answered in constructing these foundations. Later, the specifics of each experiment — and the results of those experiments — are described in their own chapter, chapter 7, answering the final three research questions.

Finally, the encoding of behaviours and model properties as aspects yields novel research approaches, a discussion of which is omitted in all relevant literature reviewed to date. Therefore, following the lead of Marsh [Mar94a] whose thesis contributed an exploration of the work enabled by their other contributions, we investigate the possibilities aspect-oriented modelling enables in chapter 8. A brief conclusion to the thesis follows in chapter 9.

Chapter 2

Relevant Literature

Include a review of the survey in Masuhara and Kiczales [MK03].

This thesis presents an aspect-oriented approach to simulation & modelling and to experiment design, with tooling to support these endeavours and an empirical assessment of the paradigm's application in the domain of simulation & modelling. In particular, the presented research makes use of aspects to augment models. The research presented in later chapters is motivated in this chapter through a review of the related literature.

Research related to this work comes from a variety of fields which do not overlap significantly, meaning that this review must cover segments of several partly related fields. The presented material is eclectic in nature as a result.

FINISH THIS AFTER EDITING / REARRANGING THE LIT REVIEW The review begins with an overview of aspect-oriented programming's design philosophy

2.1 Aspect-Oriented Programming

2.1.1 Motivations & Philosophy Underlying Aspect Orientation

Modularity is considered a key trait of maintainable, flexible, and legible programs [Par72; Dij68]. Modern modular design techniques are often concerned with segmenting a program's logic or data into separate units. The standard approach to designing with modularity in mind in many industrially-relevant languages today is object-oriented programming, where data structures and selected program logic are modularised as classes.

Some concepts in software engineering are not easily modularised by techniques such as object orientation. Guarding against unsafe concurrency usage, manual memory management, and logging are all examples of program components which are not readily modularised using an object-oriented or similar approach, because they occur alongside the logic fulfilling a program's requirements. These parts of a program are common across many modules; however, this common logic does not form traditional modules. Such concerns of a program are termed "cross-cutting concerns" in aspect-oriented programming parlance. Programmers looking to separate cross-cutting concerns into separate modules seek to address two problems they give rise to [Kic+97]:

① *"Tangling"*, where program logic essential for a program's intended purpose is intermixed with ancillary code addressing cross-cutting concerns, thereby making essential logic more difficult to maintain;

② *"Scattering"*, where program logic for cross-cutting concerns is strewn throughout a codebase, making maintenance of this code more difficult.

The existence of cross-cutting concerns is expected to make maintenance of both ancillary program logic and a program's core logic more difficult because of these traits. To address them, Kiczales et al. [Kic+97] introduced the notion of aspect-oriented programming. The paradigm is simple to define through its unique software concepts:

→ A *"join point"* defines some point in a program's execution (usually the moment of invocation or return of some function or method) where additional logic is required.

→ *"Advice"* implements some behaviour such as logging, which which can conceptually happen anywhere in program execution (i.e. what's defined would typically represent behaviour which cuts across many parts of a codebase).

→ An *"aspect"* is constructed by composing this advice with *"point cuts"*: sets of join points that define all moments in program execution where associated advice is intended to be invoked.

→ An *"aspect weaver"* then adds the functionality defined by each aspect by adding the functionality defined by its advice at each join point defined by its point cut.

The implementation of join points, and advice, or weaving is a matter left for aspect orientation frameworks and aspect-oriented languages to define. In employing the technique, aspect-oriented programming aims to separate cross-cutting concerns into aspects, removing the aforementioned repetitive code from the logic implementing a program's functional behaviour so that additional pieces of functionality — logging, authentication, and so on — can be maintained in only one place in a codebase (thereby simplifying their maintenance and comprehension), and remaining program logic can be understood and maintained without the overhead imposed by the previously tangled cross-cutting concerns.

Kiczales et al. [Kic+97] see these engineering concepts as universal throughout business logic, motivating the aspect-oriented approach for the first time. The authors present an implementation of aspect-oriented programming in Lisp, and compare implementations by way of SLOC count in an emitted C program to a comparable, non-AOP implementation, with two examples (its use in image processing and document processing). They find the idea — which they note is "young" and note many areas where research might help it to grow — can successfully separate systemic implementation concerns such as memory management in a way that reduces program bloat and simplifies implementation. It is noted that measuring the benefits of their approach quantitatively is challenging.

Tooling followed the theoretical work presented by Kiczales et al. [Kic+97] with a demonstration and subsequent technical description of AspectJ, a Java extension for aspect oriented programming [Hil+00; Kic+01]. AspectJ was introduced to satisfy the research community's need for a tool with which to demonstrate the aspect-oriented paradigm in case studies. The tool is intended to serve as "the basis of an empirical assessment of aspect-oriented programming" [Kic+01]. The library makes use of standard aspect-oriented concepts: Pointcuts, Join Points, and Advice, bundled together in Aspects. They define "dynamic" and "static" cross-cutting, by which they refer to join points at specific points in the execution of a program, and join points describing specific types whose functionality is to be altered in some way. Their paper describes only "dynamic" cross-cutting, but presents tooling support, architectural detail of its implementation, and the representation & definition of pointcuts in AspectJ. AspectJ is compared to other tools for aspect orientation and related decomposition-focused paradigms, and the authors are explicit about their approach being distinct from metaprogramming in, for example, Smalltalk or Clojure.

Filman, Friedman, and Norvig [FFN00] isolate properties of aspect orientation which they assert are definitive of the paradigm. Specifically, they claim that aspect-oriented programming should be considered in terms of "quantification" and "obliviousness":

> AOP can be understood as the desire to make quantified statements about the behaviour of programs, and to have these quantifications hold over programs written by oblivious programmers. [...] We want to be able to say, "This code realises his concern. Execute it whenever these circumstances hold."

These concepts became core concepts in aspect orientation literature alongside "tangling" and "scattering". Filman, Friedman, and Norvig give no concrete definition of the terms in their original paper and cite no sources for the terms. This thesis therefore adopts the definitions:

Quantification: the specification of points in a program in which that program should change;

Obliviousness: the property of a codebase that it contains no lexical or conceptual reference to advice which might be applied to it, and of the programmer of a target program that their code may be amended by an aspect programmer.

Filman, Friedman, and Norvig write about aspect orientation as a paradigm independent of a particular application or implementation. They are therefore able to arrive at conclusions about the paradigm in the abstract and can identify concerns for future researchers in the field and design goals for developers of aspect-oriented tooling. They note:

> Better AOP systems are more oblivious. They minimise the degree to which programmers (particularly the programmers of the primary functionality) have to change their behaviour to realise the benefits of AOP. It's a really nice bumper sticker to be able to say, "Just program like you always do, and we'll be able to add the aspects later." (And change your mind downstream about your policies, and we'll painlessly transform your code for that, too.) [FFN00]

Here, Filman, Friedman, and Norvig theorise about the advantages they expect aspect orientation to bring software engineers. However, the notion that obliviousness is a straightforward advantage of aspect-oriented programming has been contested in more recent literature. This is discussed further in section 2.1.7.

Aside from the benefits of aspect-oriented programming's obliviousness, claims made by Filman, Friedman, and Norvig such as the "painless transformation" of code when a developer changes their mind during the development process — another theorised benefit of obliviousness — is incompatible with earlier writing on modularity. Yourdon and Constantine [YC79] assert:

> The more that we must know of module B in order to understand module A, the more closely connected A is to B. The fact that we must know something about another module is a priori evidence of some degree of interconnection even if the form of the interconnection is not known.

Aspect orientation's critics describe similar incompatibilities with existing best-practices [Prz10; CSS04], as well as the lack of empirical evidence for the benefits of obliviousness [Ste06]. Claims about "better" aspect-oriented systems being more oblivious should therefore be regarded as *suggestions* from the literature. While obliviousness and quantification are useful concepts in discussing research in the field, they also give context for the research community's perspective that obliviousness and quantification are design goals for those using aspect-oriented programming [SL07; Kel08; CM06] (though Leavens and Clifton [LC07] suggest they may be best applied in moderation).

Aspect-oriented programming is thus a contentious paradigm which addresses limitations of existing methods using a novel approach, but its design philosophy also attracts sceptics. More specific criticisms of aspect-oriented programming are reviewed in section 2.1.7; specific implementations of aspect-oriented programming language extensions and frameworks are reviewed first.

2.1.2 Approaches to Weaving Aspects

The design of the weavers in different aspect orientation frameworks and language extensions is pertinent to the research in this thesis, because the research introduced

To review and compare different aspect-oriented programming frameworks, it is important to differentiate their weavers. Many designs for weavers exist. Some weavers introduce aspects when code is compiled, calcifying the program's behaviour and disallowing extension at a later time. Others weave advice as a program executes. To flexibly support the needs of a developer of a scientific model, this thesis focuses on the latter type of weaver. These weavers are referred to as "dynamic".

Chitchyan and Sommerville [CS04] present a review of aspect orientation frameworks with dynamic weavers. They compare AspectWerkz, JBoss, Prose, and Nanning Aspects through the lens of the authors' prior work on dynamic reconfiguration of software systems. By comparing different implementations of dynamic weaving, Chitchyan and Sommerville contribute a categorisation of the tools' approaches:

Need citations for these technologies

① "Total hook" weaving, where aspect hooks are woven at all possible points;

② "Actual hook" weaving, where aspect hooks are woven where required;

③ "Collective" weaving, where aspects are woven directly into the executed code, "collecting the aspects and base in one unit".

As Chitchyan and Sommerville focus on software reconfiguration rather than the mechanics and design of dynamic aspect weaving, their analysis of the reviewed tools is of less relevance to the work presented in this thesis than their generalisation of dynamic weaving. However, their review discusses the trade-offs of the three approaches identified. Chitchyan and Sommerville propose that total hook weaving allows flexibility in the evolution of a software product, at the expense of the performance of that product; this contrasts collected weaving, which shifts overhead out of the codebase and into the maintenance effort. Actual hook weaving is positioned as a compromise between the two, offering the best approach for none of their criteria but never compromising so much as to offer the worst, either. This suggests merit in a tool designed to flexibly offer any weaving approach appropriate for the task at hand.

Chitchyan and Sommerville note that one could use many of the systems they describe in practice. Though the paper is an early publication in the field, no tool the authors review offers all three dynamic weaving approaches, and none offers collective weaving alongside either kind of hook weaving.

2.1.3 AspectJ

Following their seminal work on aspect orientation, Kiczales et al. [Kic+01] published an aspect-oriented programming extension to Java called AspectJ, as discussed earlier in section 2.1.1. AspectJ is still actively developed, and is currently maintained by the Eclipse organisation. It is also the most well-adopted tool for aspect-oriented programming today, and its language for specifying aspects is adopted by other tools such as Spring AOP [Sch+14]. AspectJ can weave aspects at compile time or when join points are loaded, resolving the trade-offs between collective weaving and total hook weaving by allowing developers to select the weaving method appropriate for their use-case.

The motivations behind developing AspectJ were those of aspect-oriented programming, as the paradigm lacked an implementation at the time. Later aspect orientation research efforts have since been folded into the project, such as AspectWerks' dynamic weaving mechanisms, or have been reproduced within it, such as support within the JVM [Gol+08] and late binding of aspects [GSK10]. The popularity of the framework has led it to see some industry adoption [HJ09], although its use in real-world settings has been found to introduce a trade-off between the legibility of a codebase and the ease with which that codebase can be updated [Prz18].

I feel like AspectJ should probably have a larger section here, but honestly, it's hard to know what to say about it... hopefully this is sufficient. Only revisit if there's time.

2.1.4 PROSE

Another implementation of aspect-oriented programming with dynamic weaving is PROSE [PGA02; PAG03], a library which achieves dynamic weaving by use of a Just-In-Time compiler for Java. Popovici, Gross, and Alonso motivate the project be identifying aspect orientation as a possible solution to software's increasing need for adaptivity. Mobile devices, for example, could enable a required feature by applying an aspect as a kind of hotfix, thereby adapting over time to a user's needs. Other uses of dynamic aspect orientation they identify are in the process of software development: as aspects are applied to a compiled, live product, the join points being used can be inspected by a developer to see whether the correct pointcut is used. If not, a developer could use dynamic weaving to remove a misapplied aspect, rewrite the pointcut, and weave again without recompiling and relaunching their project.

Ideally there'd be a section for Spring AOP here too as another industry-focused

Popovici, Alonso, and Gross [PAG03] indicate that some performance issues may prevent dynamic aspect orientation from being useful in production software, but that it presented opportunities in a prototyping or debugging context. The PROSE project explores dynamic weaving as it could apply in a development context, but the authors do not appear to have investigated dynamic weaving as it could apply to simulation contexts, or others where aspect-oriented software does not constitute a product.

Their observation that aspect-oriented programming could be used for the purpose of adaptation or software prototyping instead of modularisation is an example of an alternative use case for aspect orientation suggested in Steimann's critique [Ste06] discussed in section 2.1.7.

2.1.5 Nu

Rajan et al. propose a new aspect-oriented invocation mechanism, which they call "Bind" [Raj+06]. Bind's design is motivated by opportunities to improve modularity from a design perspective: Rajan et al. assert that scattering and tangling can be introduced into a codebase *after* weaving with some weaver implementations. This complicates the use of compiled aspect-oriented code, and the development and execution of unit tests on such a codebase. Bind alleviates this issue by allowing a developer to choose when aspects are applied. This new mechanism is presented as an alternative to the weaving of aspect hooks for load-time registration into target code in the style of PROSE [PGA02; PAG03] and the direct weaving of aspect invocations in the style of AspectJ [Kic+01].

In order to demonstrate Bind's approach to simplifying post-weave codebases, Rajan et al. [Raj+06] present "Nu", an aspect orientation framework written in .NET supporting Bind. Nu's design is explained and its implementation presented, which aim to promote granularity in join point specification. What results is a flexible model for aspect orientation which is demonstrated to satisfactorily emulate many other paradigms and tools, such as aspect orientation in AspectJ and subject-orientation in HyperJ.

Might not be an issue but I'm planning to drop SOP due to time constraints, so this may stick out. Might not need defin-

Dyer and Rajan [DR10] give a more thorough technical explanation of the design and implementation of the Bind mechanism and the implementation of the Nu framework. A more technical discussion is presented, in particular on implementation details including optimisation and benchmarking, largely against AspectJ. The implementation discussed is a Java implementation, rather than the .Net implementation presented by Rajan et al. [Raj+06]. Many aspect orientation frameworks are language-specific; the existence of Nu's implementation on multiple platforms highlights the work's main contribution in the design of the Bind primitive, rather than the framework itself.

It is noted that it is "very common in aspect-oriented programming research literature to provide language extensions to support new properties of aspect-like constructs". Rajan et al. look to provide similar extensions to virtual machines, noting that their mechanism is a suitable candidate to introduce aspect orientation directly in a language's virtual machine. They position the project as a general model of aspect-oriented programming which can flexibly represent a variety of existing approaches. Bind fulfils this aim by providing a single mechanism which can be used to achieve many weaving techniques; however, case studies demonstrating that Bind can be used to make aspect orientation more practically effective for software developers does not appear to have been published.

Who notes this?

2.1.6 Dynamic Weaving in Embedded Systems

Gilani and Spinczyk [GS04] observe that while there are different approaches to dynamically weaving aspects, no approach is suitable for an embedded environment. This is due to these systems' low power and limited memory. Gilani and Spinczyk propose a framework for these situations through which weavers can be assessed for their suitability in a given environment, or generated from a set of desired features.

Gilani and Spinczyk define families of weavers by grouping the the environments they can be suitably applied to, separating them in particular by their trade-off between dynamism and resource use. There is an implication that dynamism and resource use are broadly proportional, presumably because even a carefully crafted "actual hook weaver" or JIT-compiled "collective weaver" in the parlance of Chitchyan and Sommerville [CS04] carries runtime overhead by virtue of the weaving mechanism used. A "collective weaver" embeds aspect invocations directly into their join points, which eliminates the need for an intermediary hook invocation or additional compilation. The embedded systems of interest to Gilani and Spinczyk have memory in the range of `~30kb`, where these overheads could represent significant resource use.

Aspect oriented programming's criticism is sometimes that it doesn't know what it "aims to be good for" [Ste06], and the dynamic weaving of aspects may not be good for resource-constrained environments. As Gilani and Spinczyk identify a trade-off between dynamism and resource economy, a highly constrained environment would not be able to take advantage of dynamic weaving by definition. Additionally, because the anticipated benefits of aspect-oriented programs are not observed in practice and others have noted the inefficiencies introduced by dynamic weaving [DR10; CS04], the paradigm is difficult to apply to any use-case where conservative resource usage is a concern. We might observe that researchers should seek other contexts in which to apply aspect oriented programming.

Support for this observation can be found in criticisms of aspect-oriented programming from Steimann [Ste06], whose position that alternative use-cases of aspect orientation should be explored is discussed in section 2.1.7. Examples of alternative uses include Popovici, Gross, and Alonso's proposal of using aspect orientation for software prototyping, the proposal presented by Gulyás and Kozsik [GK99] that simulation & modelling is a more appropriate field, or **Cieslak_2007empty citation**'s use of aspect orientation in modelling plant growth using L-systems. These proposed uses of aspect orientation involve a different resource economy than that imposed by embedded systems. A discussion of aspect orientation's use in simulation & modelling can be found in section 2.2.

2.1.7 Criticisms of Aspect Orientation

As referenced in section 2.1.1, the design philosophy underpinning aspect-oriented programming has its detractors. The opposition to aspect orientation is of particular relevance to the work presented in this thesis, which should be understood within the context of some perceived weaknesses of the paradigm. The following chapters introduce contributions which address some criticisms of aspect-oriented programming, so broad criticisms of the paradigm — and the work published in awareness of those weaknesses — motivates some research presented in later chapters.

Constantinides, Skotiniotis, and Stoerzer published an early critique of aspect-oriented programming [CSS04] which notes similarities between the paradigm's core concepts and `GO-TO` statements. Constantinides, Skotiniotis, and Stoerzer uses the comparison to demonstrate fundamental issues with the paradigm's philosophy: use of `GO-TO` statements is widely accepted to be bad practice [Dij68], and their infamy has become an in-joke for academics and language enthusiasts [Cla73]. They note that GO–TO statements disorientate a programmer by way of "destroying their coordinate system" — referring to a developer's technique for navigating and understanding code — which results in uncertainty about both a program's flow of execution and the state of a program at different points of its flow. Constantinides, Skotiniotis, and Stoerzer uses the comparison to GO–TO statements to question whether aspect-oriented programs can have a consistent coordinate system for developers: GO–TO statements lack obliviousness as they are visible in disrupted code, wheras aspects are not represented structurally within a program due to their oblivious design. This complicates a developer's understanding of where and how flow is interrupted. They draw comparison between aspects and COME–FROM statements, an April Fools' joke where a claimed improvement over GO–TO is developed by removing the latter's structural component [Cla73], and conclude that existing engineering techniques provide similar benefits without a trade-off in program legibility. In particular, Dynamic Dispatch is identified as a preferred alternative.

Steimann makes a similar but more thorough critique of aspect-oriented programming [Ste06]. They express concern that the popularity of aspect-oriented programming — which was nearly 10 years old at time of publication — was founded on the perception that it would solve real-world engineering problems, yet no proof existed that it was effective in practice. Steimann notes that most papers are theoretical in their discussion on tooling, that examples were typically repetitive, and that the community's discussion was concerned more with what aspect orientation could be used for than whether it worked in practice. They present a comparison between aspect orientation and object-orientation, where aspect orientation's claimed properties and principles are examined in detail, and the impact on software engineering is reasoned about from a skeptical perspective. They compare the paradigms' specific claims that they both support improved modularity against classic papers on the subjects.[1]. Steimann presents a philosophical examination of aspect orientation and assesses the paradigm against its purported merits, discussing whether we should accept the claims made by the aspect-oriented programming community. Aspect orientation's promise of unprecedented modularisation is presented as unfulfilled, and Steimann reflects critically on the state of aspect orientation research at the time. However, they conclude by proposing that the aspect-oriented approach could have a legitimate alternative use-case — other than as a *general-purpose* technique for modularising cross-cutting concerns — where it could be shown to be effective empirically.

[1]In particular, they compare against literature by **parnas_1972empty citation**.

Similar sentiments are shared by Przybylek [Prz10], who examines aspect-oriented programming in the context of language designers' quest to achieve maintainable modularity in system design. They frame the design goals of aspect orientation as being to represent issues that "cannot be represented as first-class entities in the adopted language". Przybylek questions whether the modularity offered by aspect orientation can really be said to make code more modular. In particular, they distinguish between lexical separation of concerns and the separation of concerns originally discussed by Dijkstra [Dij82]. They assess principles of modularity — modular reasoning, interface design, and a decrease in coupling — and find that the aspect-oriented paradigm can detrimentally impact the expected benefits of proper modularisation in a program by "violating basic software engineering principles" such as information hiding and structured programming. They suggest that aspect orientation's claimed benefits are a myth repeated often enough to be believed.

This review is arguably unfinished — there are more papers commented below this note which are probably worth reading / ensuring I'm citing. Tim has noted that the review on early / foundational literature of AOP seems incomplete, and I think the commented papers to review below could be good candidates to address this.

2.1.8 Alternatives to Aspect Orientation

Aspect-oriented programming's goal of modularising cross-cutting concerns is shared by other paradigms. As discussed in section 2.1.1, the work first introducing aspect orientation by Kiczales et al. [Kic+97] makes note of similarities to reflection, metaprogramming, and program transformation. They also observe that other disciplines have introduced "aspectual decomposition" independently. Some modelling & decomposition techniques which are related to aspect orientation are discussed here.

System Diagrams

The example of pre-existing aspectual decomposition by way of diagramming given by Kiczales et al. [Kic+97] is in physical engineering. To give a concrete example from their description, differing types of diagrams when engineering a system such as thermal and electric diagrams of a heater are described as "aspectual" because of the modular nature of the diagrams; though there might be many diagrams of different kinds, they compose together to give an overview of the system being designed.

Similar diagramming techniques have independently arisen in other domains since. The Obashi dataflow modelling methodology[WC10] by Wallis and Cloughley models all possible paths of dataflow through "B&IT" (business and IT) diagrams, where business-specific concerns (people, locations, groups, and business processes such as payroll, stock-check or budgeting) are modelled alongside IT concerns such as applications supporting business processes and the software and hardware infrastructure supporting them. Modelling dataflows in this way allows for a comprehensive understanding of assets and business processes. However, in order to understand how data flows between specific assets within a B&IT, sub-graphs ("DAVs", or Dataflow Analysis Views) denote specific pathways through which data flows between source and sink assets. Alternatively, a B&IT can be viewed as a composition of all possible DAVs within an organisation. Dataflows are therefore broken into different diagramming techniques and specific business concerns can be described independently of others, even if these concerns interact in their dataflow pathways (and, therefore, cutting across each other). Obashi therefore allows for the aspectual decomposition of business processes, through the description of an organisation by individual dataflow analysis views, which compose into an overall model of a system in a B&IT diagram. Obashi models are an instance of aspect orientation which were designed for simplicity and comprehension[WC10; Seo11], but trade this for domain-specificity.

Other diagrams of systems, called "bigraphs", were produced by Milner [Mil08]. Bigraphs are a category-theoretic process calculus able to simulate petri nets, π-calculus, and λ-calculus [Mil09] but are also used in the simulation & modelling of real-world systems such as networks [Cal+14] and ubiquitous computing [Ben+16]. A key property of bigraphs are their "ports", which denote possible points of composition a given model can have with others. Bigraphs' compositional property is reminiscent of the use of aspect-orientation for composition [**Cieslak_2007**; GK99]. While the existence of a port means bigraphs cannot be composed obliviously, they are a powerful tool for formal modelling and are used in a similar manner.

Metaprogramming

Metaprogramming is identified as a precursor to aspect orientation by the original paper on the paradigm [Kic+97]. Research into the use of metaprogramming to simplify the composition of software modules was undertaken at a similar time by Keller and Hölzle [KH98], who introduce a technique they name "Binary Component Adaptation" (BCA). Their research into BCA the difficulties involved in the integration of software components, particularly considering their evolution over time where

Wouldn't the paragraph below fit better in the section on program mutation?

Reply/Q: I'm not sure exactly what section this is, but have components are re-used with differing requirements. By modifying binaries directly, incompatibilities between a program and one of that program's dependencies can be resolved by way of mutating either after compilation. Their implementation defines a representation for the modification of pre-existing Java class binaries, the output of which can be verified as also being valid Java class binaries. Keller and Hölzle claim that BCA allows for dynamic modification of programs with little overhead. They believe BCA is unique in its combination of features, which include engineering concerns such as typechecking code which is subject to adaptation and its obliviousness to source implementation, as well as guarantees that modifications are valid even for later iterations of the program subject to adaptation.[2]

Metaobject protocols describe the properties of an object's class (including, for example, its position within a class hierarchy) in an adaptable manner [KDB91]. Espák [Esp04] note that this technique can be used to implement aspect orientation, therefore providing at a minimum the same functionality, though they achieve this through reflective programming techniques and are designed with metaprogramming as a primary goal as opposed to modularisation of cross-cutting concerns [KDB91; Sul01].

Subject-Oriented Programming would go here, but I haven't written it up and I'm running out of time. I don't think it's the most essential thing to include — it's relevant to AOP's history but not really to the work I'm presenting in later chapters. Write up if there's time...

Engineering Techniques with Related Aims

Multiple-dispatch, where methods on objects are chosen to be run based on the properties of the parameters passed at point of invocation, allows for oblivious decomposition without the need for a weaver [DGM08], although this does not support the goals of aspect orientation in totality. For example, a programmer might want their program to exhibit differing behaviour when methods are called with differently-typed arguments, which is supported by multiple dispatch. However, they might instead want their program to exhibit some additional behaviour whenever a method is invoked, such as logging, but might not want to implement logging alongside the rest of their method implementation for clarity or length reasons. Multiple dispatch therefore offers comparable but different functionality to a software engineer.

Engineering patterns such as decorators provide similar functionality to aspects [FBS17], in that cross-cutting concerns can be separated into their own module, but they differ in their approach

[2] BCA shares concepts with aspect orientation, but is also a promising technology for the introduction of process variance; see section 2.3 for a discussion.

to obliviousness: decorators annotate areas of a codebase they are applied to, and therefore do not offer obliviousness as aspects do. Decorators allow for additional functionality to be written as a separate module and applied as a wrapper around a function definition. A function with a wrapped definition is replaced by a function returned by the decorator, which takes the original definition as an argument [Gam+95]. Additional logic can therefore be simply applied before or after a function. While this achieves a similar effect to aspect orientation — as logic is added before and/or after the original logic of a function — its design principles are different as annotating the function's definition directly marks it as altered, and so the original definition cannot be oblivious to the change.

A diagram or Python example might help here, as per Tim's suggestion below. Add one if there's time.

2.2 Aspect Orientation in Simulation & Modelling

Would some illustrative examples of these different approaches help?

Reply/Q: Yes, I'm sure they'd help! Diagrams to add here if there's time.

This thesis is concerned with the use of aspect orientation in simulation & modelling codebases; it is therefore necessary to review related research in simulation and modelling. This section reviews literature contributing aspect-oriented tooling and the philosophy of aspect-oriented experiment design in the simulation & modelling community.

2.2.1 Suitability of Aspect Orientation in Experimental Codebases

Gulyás and Kozsik [GK99] observed that, in the study of complex systems through software models, the codebase produced serves two purposes: the experimental subject, and the observational apparatus used to conduct the experiment itself. They use this framing to identify that the observation of a program's state constitutes a cross-cutting concern. In order to reduce scattering and tangling in experimental codebases, Gulyás and Kozsik theorise that aspect-oriented programming may separate the logic of observation from the core logic of a simulation or model.

The approach is demonstrated via their Multi-Agent Modelling Language (MAML). MAML was designed to enable aspect-oriented simulation of agent-based models and was implemented using the SWARM simulation system [Hie+94], a domain-agnostic framework for agent-based simulation. While SWARM takes the form of a collection of C libraries, MAML is implemented as a domain-specific modelling language, allowing it to support aspect-oriented programming as a language feature.

MAML's aspect orientation effectively makes use of Observer patterns to measure a simulation's state. This enables a researcher to observe an experiment without the necessary logic being scattered or tangled in their domain model. Gulyás and Kozsik find that aspect-oriented programming provides an intuitive and straightforward method by which simulated experimental systems can be composed. They note that MAML's simplicity and its philosophy on modelling are more "satisfactory" than Swarm's standard approach. The team report that MAML's implementation was more complex than initially conceived: the `patch` unix tool was intended for use as their weaver, though the team eventually developed a transpiler from MAML to Swarm instead (which they name `xmc`.). The deciding factors for the development of a custom transpiler are not discussed.

In addition to presenting tooling for aspect oriented simulation, Gulyás and Kozsik [GK99] theorise about the potential benefit of applying aspect orientation to simulation & modelling. They observe that there may be benefits beyond improvements to modularity and a reduction of tangling & scattering. In particular, their work discusses specific scenarios in which the *type* of separation of concerns offered by aspect orientation is desirable, and the engineering approach to achieving the aim reasonable. This distinguishes the work in comparison to most other research on aspect orientation. Many papers describe the expected benefits by simply drawing from existing literature and the claims made in Kiczales et al. [Kic+97]'s first paper on the subject, rather than drawing on empirical evaluation [Ste06; Prz10; Prz18] as discussed in section 2.1.7.

T: If there's time when redrafting, note which scenarios they distinguish between specifically.

This is a rare example of philosophy on aspect oriented programming's suitability in a particular use-case. Gulyás and Kozsik's work is of particular importance in this review, because the domain they identify as particularly suitable is simulation & modelling, which is the subject of this thesis. That aspect orientation might be well suited to separating observer and experiment partially motivates research in later chapters, which investigates the use of aspects to *modify* simulated behaviour rather than simply observing it. Further discussion follows in section 2.4.

2.2.2 Aspect Orientation in Discrete Event Simulation

Aspect-Orientated Implementations of Simulation Frameworks

Chibani, Belattar, and Bourouis [CBB13] observed that simulation frameworks and the codebases built upon them can exhibit cross-cutting concerns such as event handling, resource sharing, and the restoring the state of a simulation run. They investigated the introduction of aspect orientation to Discrete Event Simulation (DES) frameworks to address tangling and scattering that may arise from the cross-cutting concerns they identified. They contribute a discussion of aspect-oriented programming's potential application to DES codebases, and detail the avenues available for research in the field. Chibani, Belattar, and Bourouis [CBB13] identify Japrosim [AB08; BB14] — a DES framework previously developed by the research team, which was designed to model domains through process interaction in Java — as an example of an existing framework which exhibits the cross-cutting concerns they describe.

Later, Chibani, Belattar, and Bourouis [CBB19] identified opportunities for the use of aspect orientation in simulation tooling, aiming to increase "modularity, understandability, maintainability, reusability, and testability" by applying the paradigm [CBB19]. They present a case study of an application of aspect orientation to simulation tooling by identifying cross-cutting concerns in Japrosim, a discrete event simulation framework, and propose an aspect-oriented redesign of the tool using AspectJ. Chibani, Belattar, and Bourouis describe Japrosim's existing object-oriented design, followed by aspect oriented variations of some design elements, including concurrent process management and in Japrosim's graphical animation features. A similar survey of areas in which Japrosim's source might benefit from the application of aspect orientation is presented by Chibani, Belattar, and Bourouis in an earlier work [CBB14]. In both cases, the main contribution noted is the design itself. Evaluating the main improvements between the presented aspect-oriented design and the existing object-oriented one is left to future work.

Chibani, Belattar, and Bourouis [CBB19]'s later work presented an implementation of their proposal and provided a quantitative evaluation of that implementation. The quantitative evaluation provides measurements based on Martin [Mar94b]'s object-oriented design metrics and demonstrates a greater independence of packages in their aspect oriented version of Japrosim than in the original. However, the intended aim of aspect orientation is not to decouple existing packages, but to isolate those packages' cross-cutting concerns into new ones. It is therefore unclear that their quantitative evaluation achieves its improvements as a result of aspect orientation. No further discussion of their results is provided, and it is possible that the improvement is due to the rewriting necessary in their maintenance of the Japrosim source, rather than due to their use of aspect orientation specifically.

Similarly to Chibani, Belattar, and Bourouis [CBB19; CBB13; CBB14], Aksu, Belet, and Zdemir observe that there are advantages in adopting aspect orientation when developing a simulation framework [ABZ08]. Examining the DES framework Simkit [Gom+95], they motivate two different applications of aspect orientation: a refactoring of the framework itself to better manage cross-cutting concerns within its codebase; and aspect-oriented tooling for use by modellers who represent cross-cutting concerns within their models. Opportunities for improvements in production and development are discussed, and some implementation notes are detailed, although no concrete implementation or evaluation is provided; the work instead proposes design alterations, and the authors "leave it as a future work *[sic]* to explore the usability and efficiency" of aspect orientation used idiomatically alongside Java's existing reflection offerings. The existence of multiple attempts to refactor differing simulation packages with aspect orientation indicates potential for modellers in the use of aspect-oriented patterns, but the real-world utility of the techniques are omitted. As is common in aspect orientation literature, Chibani, Belattar, and Bourouis and Aksu, Belet, and Zdemir both defer to the general claims that aspect orientation improves modularity of cross-cutting concerns and can eliminate code smells such as tangling and scattering. Chibani, Belattar, and Bourouis do present some quantitative evaluation, but this is flawed as previously described.

Aspect-Orientated Implementations of Simulations & Models

Research projects applying aspect orientation to the implementation of simulation frameworks [**chibani**] fails to provide a case study which evaluates their technique with real-world examples. However, case studies do exist which evaluate aspect orientation's suitability in maintaining model source code.

Ionescu et al. identify an increased demand for computational power in simulation execution on supercomputers [Ion+09]. Existing known-good models might be unsuitable for the extreme requirements of code efficiency modellers contend with, but running the code in different environments requires modifications to make the code suitable in the environment. These modifications are subject to regulations and introduce risk of a reduction in quality during maintenance. The authors propose an aspect-oriented solution to the problem, where aspects modify the simulation codebase with minimal overhead. An implementation of a real-world model for disaster prevention is presented, and assessed both by comparison against an equivalent non-aspect-oriented codebase and by assessment of the aspect-oriented variant's scalability and reliability in both cluster and multi-cluster environments. They find that a comparative analysis of generated code and of their simulations in various configurations both indicate that their simulation's aspect-oriented implementation is suitable for use in disaster prevention, implying that aspect orientation could be suitable in scenarios with comparable requirements.

This work provides evidence that aspect-oriented programming is suitable for supporting modifications to simulations and models. This contrasts the lack of evidence for the paradigm's success in improvements to modularity and maintainability [Prz10; CSS04], and supports Steimann [Ste06]'s suggestion that aspect orientation might be suitable for other uses — as well as the suggestion made by Gulyás and Kozsik [GK99] that the paradigm is well suited to the requirements of simulation & modelling codebases.

As it is a rare example of aspect-oriented case studies, the methodology employed by Ionescu et al. is important to highlight. Their evaluation measures code quality improvements, which are claimed using similar measurements in early aspect orientation research [Kic+97]. Their code analysis makes use of significant lines of code as a core metric, which doesn't reliably reflect code quality. As Rosenberg [Ros97] explains:

> […]the best use of SLOC is not as a predictor of quality itself (for such a prediction would simply reduce to a claim about size, not quality), but rather as a covariate adjusting for size in using another metric.

Although Ionescu et al. [Ion+09] evaluate code quality, the methodology employed to measure improvements is unreliable.

Improvements in code quality are those which have come under scrutiny by the critical papers reviewed in section 2.1.7. The results presented by Ionescu et al. do not satisfy critics' requests for empirical evidence of improved code quality. This does not impact their aspect-oriented models' viability: their study demonstrates that their models can be augmented to support new supercomputing environments without lack of performance. The models described in this work satisfy that aim: their models are also evaluated in their performance. Model performance is a priority in supercomputing contexts, where execution time is financially expensive and energy-intensive. Quantitative evaluation of their models' execution time shows less than 5% slowdown compared to a non-aspect-oriented implementation. Ionescu et al. deem this a reasonable trade-off for the engineering improvements they observe.

Ionescu et al.'s application of aspect orientation to supercomputing & disaster prevention simulations meet their performance requirements and demonstrate a modelling technique which adapts existing models for use in new environments without directly modifying pre-existing source code. This result is notable with regards the contributions presented in this thesis, which similarly aim to augment a pre-existing model without directly modifying its source code — though this thesis' targets model reuse and design simplification rather than for avoiding the regulatory overhead and financial cost of maintaining models which run on supercomputers.

2.2.3 Aspect-oriented L-Systems

Aspect orientation is also applied in other simulation paradigms. Cieslak et al. [Cie+11] investigated the use of aspect orientation in L-system based simulations. An L-system[Lin68] is defined by a set of symbols, an initial string composed of these symbols, and a set of rules for rewriting substrings. While being a powerful tool for representing fractal structures, they were originally conceived of for modelling in botanical research.

Cieslak et al. note that some details of plant modelling are actually cross-cutting concerns against many plants or families of plants, such as carbon dynamics, apical dominance and biomechanics. To represent these, they introduce a new language to describe plant models which makes use of aspect orientation to represent these cross-cutting concerns. They successfully test the approach by representing carbon dynamics, apical dominance and biomechanics as cross-cutting concerns that are integrated into a previously published model of kiwifruit shoot development. Cieslak et al. hope that these cross-cutting concerns might work in other models too, but this is untested. The use of an aspect developed for use in one model and reused in another seems untested in the community's literature and is a noted omission in the conclusion of this particular work.

2.2.4 Aspect Orientation & Business Process Modelling

Several projects within the business process modelling research community make use of aspect orientation to design modelling languages which produce less monolithic business process models [Cap+09; Sil+20] and simplify the composition of models [CM07]. Business process modelling research is relevant to this thesis' contributions, as business processes are inherently socio-technical and later chapters present tooling for and results in the modelling and simulation of socio-technical systems using aspect-oriented techniques. In addition, some research conducted prior to this thesis developed software engineering processes that are conceptually similar to business process modelling (see chapter 3). There also exists interest in modelling behavioural variance within the business process modelling community (see section 2.3), which is relevant to this thesis' concern with the aspect-oriented representation of changes to processes and modelled behaviours.

Charfi and Mezini observed opportunities integrating BPEL, an executable business-process modelling language, with aspect-oriented concepts [CM07]. This is because when BPEL systems are composed together the logic being composed can lack the flexibility required by BPEL's use cases. The specific use-case examined is web service definition, where changes affecting composition of multiple component parts can affect many areas of a final result, making modification error-prone. The authors specifically seek to support dynamic workflow definitions — "adaptive workflows" — which BPEL's existing extension mechanisms do not sufficiently support but which aspect orientation literature does discuss (see the aspect-oriented programming implementations discussed in section 2.1). Therefore, they look to construct an aspect-oriented BPEL extension. Using the case study of modelling a travel agency's web services, they create an aspect-oriented extension by first defining how such an extension would be represented graphically in BPEL's workflow diagrams. Further detail is added to arrive at a technical definition with XML representations, weaving mechanics, and eventually the construction of a BPEL dialect, AO4BPEL. The authors find that their pointcut system (which describes join points on both processes and BPEL messages), support for adaptive workflows, and aspect-oriented approach to workflow process modelling make AO4BPEL unique at the time of publication, though related AOP implementations exist in each individual area of their contributions. The work is weakened by brittle semantics around pointcuts, join points, and the temporal nature of workflow modelling. For example, they note that defining contingent behaviour — only applying an aspect conditionally, based on a trace through a simulation of a modelled system — would allow weaving advice only when model state deems this appropriate. They also call for more generally theoretical AOP research, which mirrors the requests some critics of aspect orientation research make (as noted in section 2.1.7). The contingent application of model adaptation is a motivating case for some work presented in this thesis; see chapter 3 for a discussion.

Charfi and Mezini [CM06] described AO4BPEL in detail and presented a generalisation of the notation developed for it which applies to any graphical workflow modelling language. Accompanying this are some examples of its use building a framework for enforcing certain requirements of BPEL models, and use of that framework to develop aspect-oriented frameworks for enforcing security and reliability within AO4BPEL models.

In later work, Charfi, Müller, and Mezini defined a similar aspect-oriented dialect of BPMN they name AO4BPMN [CMM10], after asserting that the concerns addressed by AO4BPEL [CM06; CM07] in the field of executable process languages also apply to business-process modelling languages, and can be solved similarly. The generalised notation of aspectual workflow models presented in Charfi and Mezini's thesis [CM06] are applied to BPMN to produce an aspect-oriented language specifically for process modelling, as opposed to executable business process modelling.

can you say something explicit about the difference and benefits of this?

Reply/Q: I don't think I can off the top of my head right now, but I'm leaving this here to consider more if

Cappelli et al. also note that cross-cutting concerns exist in business process models, and are specifically motivated by monolithic design approaches common in business process modelling languages. Like Kiczales et al., they claim that a lack of modularity in business process models leads to cross-cutting concerns scattered throughout a model [Cap+09]. To alleviate the issue, they propose a meta-language, AOPML, which incorporates aspect orientation in a metamodel of business process modelling languages, and instantiate it within their own dialect of BPMN. Using a model of a steering committee as a case study, and separating cross-cutting concerns such as logging, the paper proposes reducing complexity and repetition graphically, thereby in a manner more in keeping with the language design philosophies of popular business process modelling languages, the design and use of which are typically graphical [Obj11; DOR95; Obj15]. They note that this is in contrast to other applications of aspect orientation in business process modelling — specifically AO4BPMN — where aspect definitions written in XML concern not only the advice to be applied but also their relevant join points, as in general programming aspect orientation implementations such as AspectJ. In this way, AOPML exhibits the spirit of business process modeling more stringently than does Charfi and Mezini's notation for aspect-oriented workflow modelling.

Can you say something more precise than spirit here, e.g. objectives, intentions at least?

Reply/Q: Yes, but I kind of like this phrasing! I want to get at the community / idioms as well as the technical intent. Leaving this here to reconsider if there's time later on.

The difference between Charfi, Müller, and Mezini's approach in designing AO4BPMN [CMM10] and Cappelli et al.'s approach in designing AOPML [Cap+09] highlights design decisions taken when introducing aspect orientation in a new domain. There is an opportunity for a domain-specific aspect orientation framework to align its design with the traditions and idioms already present in models within that domain, but doing so may break the traditions and idioms which already exist in aspect-oriented approaches in other domains. Comparing the approaches of Charfi, Müller, and Mezini and Cappelli et al. demonstrates that there may be no clear "best" design approach when blending pre-existing modelling paradigms, such as the graphical modelling languages used in business-process modelling and the abstract concepts of aspect orientation. The discussion around whether it is more desirable to adapt existing design elements of aspect-oriented frameworks to a given domain or adapt that domain's existing modelling traditions, idioms, conventions or syntax to incorporate aspect orientation as it is used elsewhere is outside the scope of this thesis.

New concepts within the design of aspect orientation frameworks are addressed in the business process modelling community. Jalali, Wohed, and Ouyang note that aspect oriented modelling frameworks often do not explicitly model the precedence of aspect application [JWO12]. They address this limitation by defining a mechanism to be used in capturing multiple concerns as aspects, where the invocation of advice must follow a certain precedence. The aim of the work is not to propose tooling around the precedence of aspect application so much as to contribute to aspect oriented design theory, providing a notation for precedence which is broadly applicable. The precedence model is, put simply, that a mapping exists for each application of advice to join point such that the mapping defines an ordering on advice for that join point. The definition defines "AOBPMN", a formalised dialect of BPMN supporting aspect orientation with precedence. A case study is provided where AOBPMN is instantiated within a coloured Petri net. Their study expands on existing work by research teams led by Capelli [Cap+09; Sil+20] and by Charfi [CM07], in that it develops a mature formalism for and model of aspect orientation as applied to business process modelling. However, Jalali, Wohed, and Ouyang note that their case study is limited in scale. No tooling or evaluation of the practical benefit of their approach is provided.

add something early on giving an example of how it might be a problem?

2.3 Process Variance in Simulation & Data Generation

Research presented in later chapters concerns the application of aspect-oriented programming to the codebase of a model to represent changes within it. In particular, we are interested in representing changes to simulated behaviour. This section reviews techniques where variations on already-modelled behaviour are represented in some way.

2.3.1 Discussion of Variation & Motivations for Variations in Process Models

Difficulties arise when obtaining real-world datasets for many reasons. For example, large empirical datasets are typically produced by organisations which would prefer some level of secrecy around their operations, making publishing those operations for the investigation of research teams unlikely. Researchers collecting these datasets describe a "lengthy process" [Don20] and explain that traces of real-world processes are hard to obtain because "higher management [can be] worried about the risks" of publishing such datasets [Don20]. Another factor contributing to the difficulty of collecting empirical datasets is that it is often infeasible to do so, either because there is a need to study the process before implementing it (making synthetic datasets the only option available to researchers), because the process is not yet fully understood (making simulation of many variants of that process useful in aiding understanding), or because the dataset itself is of interest to researchers rather than the real-world system that produced it (such as in the case of evaluating process mining techniques [VW04; AGL98]). The generation of datasets with specific properties is its own field of research; the literature of interest specifically concerns techniques for the modification of a model or simulation to generate those datasets.

Citations for all difficulties in collecting empirical datasets would be nice...

The reviewed techniques allow a programmer to include some (separately defined) modification of a process, or represent the modifications of a varied process within simulated output [SA13; SA14]. The benefit of this is that possible changes to a process can be described once and applied to that process where appropriate, resulting in datasets which are affected by the modification. Changes to processes might describe attempts to circumvent security protocols, laziness or confusion in a human actor within the model, or random “noise” to produce synthetic log traces containing aberrations which mimic those found in noisy empirical datasets. In all cases, behavioural variations can be described as some alteration to a process and applied to either a model or the product of that model (datasets or log traces) to represent the same alteration introduced at an arbitrary point in the simulation.

This decouples the expected behaviour encoded in the original model from simulated behaviour, which is obtained by composing the model and behavioural variation using a given technique’s method for doing so. This approach to modelling behavioural variation allows the same altered behaviour, which would otherwise be described in many disparate points in a model, to instead be written once and introduced wherever required. The observation that the same variation might appear in many areas of a model, and that the variation can be separated from the model and introduced where necessary, frames the modelling of these variations in the same way as aspect orientation frames cross-cutting concerns. The work presented in this thesis explicitly applies changes to processes and simulated behaviour as aspects in the same manner. Therefore, although this aspectual connection is not made explicit in much of the literature to date, it is important to review literature on simulation and modelling which modularises these variations; this section reviews that literature. The work reviewed is highly relevant to the contributions in this thesis, in particular because the core motivations of this field are shared by the research presented in later chapters. For this reason, the section leads by discussing those motivations and their relationship to aspect orientation in detail.

2.3.2 SecSY

Research undertaken by Stocker and Accorsi [SA13] aims to synthesise process logs which are representative of attackers’ efforts to compromise the security of a modelled system. Their research project, named “SecSY”, is an attempt to address issues arising from the difficulty of retrieving representative log traces for security-critical systems in which attacker activity is present. Logs are developed by process simulation through "well-structured" models, a mathematical property on which transformations were previously defined by Vanhatalo, Völzer, and Koehler [VVK09]. The authors develop a tool for the simulation of a process using well-structured process models, and apply transformations to both the model before execution and the log it produces through the trace of a simulation.

They conclude that their tool is performant and verify it can produce logs representing security violations by way of analysis through PROM, a popular framework for process mining, and pre-defined security constraints on their models. They also note some limitations: log traces cannot be interleaved (due to a lack of parallel simulation of processes), may be incomplete (missing violations), and that mutated models and traces are not guaranteed to be sound by construction. However, they see their proposal as a necessary step in realistic data generation for business processes. A further weakness of the work is that model and trace modification techniques are limited: processes can be added or removed, but complex graph transformations are only permissible when representable through the composition of the mutation primitives they provide, on which there are only three for processes: swapping AND and XOR definitions of process gateways, and swapping process order. Mutations cannot be applied contingent on the state of a simulation run, for example, representing a decision taken by an attacker based on what had already happened.

In later work, Stocker and Accorsi detail the technical aspects of SecSY, their tool for implementing the generation of synthetic logs which use their technique [SA13] to represent security violations of security-critical business processes. A Java implementation of SecSY is described, which simulates well-structured models and applies mathematically-defined transformations on the model being simulated (before simulation occurs) and the logs obtained through simulation traces. An improvement on earlier work is that custom transformers can be written. However, a limitation of the original work remains, which is that users cannot easily dictate the degree to which variations are applied.

2.3.3 Cross-Organisational Process Mining

Pourmasoumi et al. also address the need for access to variations on business processes, though for the development of a research field, "cross-organisational process mining" [Pou+15]. Process mining can require many process logs, as does the benchmarking and evaluation of process mining techniques.

Traces from business processes which are similar but not identical can produce log traces which reflect that similarity, but also reflect the variations in different instances of those processes. These log traces exhibiting variation can be used in the training and analysis of process mining tooling and techniques, which must contend with natural variation present in the execution of real-world traces. To support the field, log trace generation from a variety of process models is therefore required. Such logs are not in adequate supply, as explained in section 2.3. The authors' approach to the problem is to present an algorithm for the mutation of business processes, such that simulation against variations of the business process can produce process logs reflecting those variations. Their algorithm makes use of structure tree representations of processes, which models processes as trees and permits conversion to BPMN models and Petri nets [Bui14]. Pourmasoumi et al. make use of this constraint to demonstrate that their models are block-structured, a mathematical constraint on model structure which 95% of models have been shown to comply with [Li10]. Their contribution is a set of transformations on structure trees and block-structured models, and an algorithm applying these transformations to process models, and a tool which implements it built on PLG, a process log generation tool. They conclude that tools such theirs can be used to generate log traces representing process variation, in such a manner as to satisfy the requirements of the process mining research community.

Pourmasoumi et al. describe a list of model transformations they explain is "not intended to be comprehensive" [Pou+15], which makes its full potential unclear. Additionally, processes their tool applies to must be block-structured. The importance of this requirement is that it limits their model transformation technique to business process models. It is not demonstrated that models of processes in other domains satisfy the condition, such as the flow of data [WC10] or behaviour of human or technical actors in socio-technical systems [WS18a]. Finally, the tool is limited by its lack of capacity to represent variations which are applied contingent on system state. A use case for modelling behavioural variance is to model changes which are impossible to anticipate from the vantage of a modeller, such as a socio-technical system's human actors' mistakes, security exceptions and violations, or corrective actions to mitigate undesirable system state. Pourmasoumi et al.'s tool produces variations on a process model, but modelling behaviour which is expected to vary within iterations of the *same* model is outwith the scope of their project.

2.3.4 Executable BPMN Models

Mitsyuk et al. are motivated by the field of process mining's requirement for datasets of process logs made from well-understood process models, defined in a high-level manner [Mit+17]. They demonstrate a technique for generating event logs from BPMN models by introducing algorithms for the direct simulation of BPMN models and the collection of traces from those simulations. While their approach does not support the simulation of all BPMN concepts, notably message passing, they provide a tool which produces log traces for a BPMN model through PROM, a standard tool within the process mining community [Van+05]. This results in their technique providing high-level model simulation through already-standard tooling, meaning adopters of the technique need not rely on dedicated tooling which may not be compatible with other researchers' process mining techniques.

The algorithms presented by Mitsyuk et al. simulate processes described by BPMN models, but don't include any provisions for representing variance. However, the technique could be combined with aspect orientation techniques for BPMN as discussed in **??** [CMM10; Cap+09] to represent alternate behaviour applied contingently. Demonstrating the viability of this approach is an avenue of research beyond the scope of this thesis. However, the motivation of the work mirrors that of other research projects reviewed in this section: a need for synthetic datasets of traces through a process, for use in scenarios where empirical datasets are difficult to obtain.

2.3.5 BPMN Extensions for Process Variation

Machado et al. note that there are operational costs to the inefficient modelling of business processes. Specifically, they note that costs can be replicated across automated processes, and failure to identify such scenarios give rise to these operational costs. This work's core motivation is that the representation of variation in process models would allow for the capture of a replicated process once, with instances of similar processes described as deviations from that captured blueprint. On this basis, Machado et al. extend BPMN to support the notion of individual processes as transformations of an underlying process, i.e. that a given process can sometimes be expressed as a deviation from some pattern, and is therefore define-able as the composition of that pattern and a variation upon it. Their approach is illustrated through two small, broadly similar business processes initially modelled in BPMN and then represented in Haskell, allowing the authors to demonstrate their representation of variability as process deviation with realistic examples.

While the work presented makes no empirical evaluation of their technique, Machado et al. note that their industrial partner responded positively to the research presented in this publication and that further technical improvements are to be made (support for around advice, and for quantification). They also express an intent to conduct a real-world evaluation in the HR domain. While we are unaware of any real-world evaluation undertaken by this research time to date, some formal proofs that the transformations their tool supports are always well-formed have been contributed [MAG12].

Review Mitsyuk and Shugurov [MS16] and other work with Shugurov if there's time…

2.4 Discussion

This thesis draws on the idea that, in response to Steimann asking what aspect orientation is good for, Gulyás and Kozsik would seem to answer, "simulation & modelling". This discussion concludes the literature review by summarising the observations made and motivating the work presented in future chapters.

Worth observing that Ionescu et al. [Ion+09] actually produce a case studdy of using AOP and their code quality evaluation uses SLOC (unreliable), but they do find that they can augment models successfully (in their context, successfully is without significant

Aspect-oriented programming is designed to permit highly modular software engineering in scenarios where cross-cutting concerns are identified, by isolating them as separate modules [Kic+97]. The aim in employing aspect oriented programming is to reduce tangling, where a cross-cutting concern is intertwined within a program's main concern, and scattering, where the same cross-cutting concern is re-written at many points in a program's source. Aspects which modularise that concern can be written once, separately from a program's main concern, and re-introduced to each point in a program to which the aspect relates by way of weaving. An aspect orientation framework must therefore be able to quantify the points at which the aspect applies [FFN00]. Aspects specify both advice (the implementation of its associated cross-cutting concern) and the join point defining where in a target program that advice should be woven. Aspect orientation therefore implies that the source aspects are applied to are oblivious to their application [FFN00].

The design of the paradigm is intended to increase modularity in the software applying it [Kic+97; FFN00; Prz18], and its proponents often claim this modularity as a benefit of the aspect-oriented approaches of their research [GS04; CM07; Cap+09; JWO12; CBB13]. However, its critics question the reasoning around these benefits, and note that there is little empirical study into whether aspect oriented programs truly benefit as a result of this modularity [CSS04; Ste06; Prz10; Prz18].

One appropriate application area may be in the representation of behavioural variation in simulation & modelling. The application of aspects to models is already well-studied [ABZ08; CBB13], particularly within aspect-oriented business process modelling [CM07; Cap+09; JWO12], where modelling behavioural variation has also seen some prior research [Mac+11; SA13; Pou+15; Mit+17]. Outside of business process modelling, aspect orientation would reasonably be expected to support the development and observation of models themselves [GK99].

Research opportunities at the intersection of aspect orientation and behavioural variation in modelling occur because behavioural variation is an example of a cross-cutting concern. Changes to expected behaviour such as laziness, boredom, confusion or misunderstanding can impact many parts of a process in a socio-technical system, but modelling the variation caused by any one of these requires similar alterations to behaviour in many disparate parts of the model they occur within. Behavioural variations are therefore both scattered and tangled, and constitute a cross-cutting concern which might be well suited to modularisation in aspects. They may also cut across models, requiring a re-implementation for each codebase if not modularised.

Existing aspect orientation techniques and behavioural variation modelling techniques are ill-equipped to take advantage of this alignment. That behaviour changes when it varies is tautological; however, changes supported by existing aspect orientation techniques weave advice before, after, or around their join points, and therefore do not alter the definition itself. In some systems, some variations may be representable as additions inserted before and/or after some other behaviours, but such techniques are unsuitable for representing modifications of the target behaviour itself, or behaviours which should be omitted instead of added. Additionally, join points available to an aspectual programmer may not be granular enough to permit representing the changes they require in such as system, and aspect orientation's principle of obliviousness opposes the modification of target code to make new join points available. Techniques which would directly rewrite target source are typically extremely low-level, and therefore ill-suited to most modelling applications [KH98]. Other techniques which permit defining changes to processes at a high level may allow a modeller to *describe* intended changes (such as the high-level annotations supported in AOPML [Cap+09]), but such techniques are intended for human interpretation, not machine execution for simulation purposes. These techniques permit describing behavioural variation within another process, but only by virtue of the flexibility of natural-language annotation, and are therefore unsuitable for simulation and modelling purposes.

Such techniques also lack executable notion of "state". Real-world behavioural variance can often be contingent on the environment an actor exists within. While variations such as security breaches might be predictable (by identifying weaknesses in existing processes, for example), variance in socio-technical systems often occurs in the behaviour of human actors. This might be in response to a degraded mode [JS07], where behaviour naturally drifts to a functional — but undesirable — state, or due to an individual making a mistake, forgetting procedure, or being in a state which alters their behaviour, such as tiredness or drunkenness [OLe20]. A framework for modelling behavioural variation using aspects should therefore apply that aspect to a simulated system contingent on the state of that system at a given point in time. High-level modelling technologies such as BPMN and OPM are executable [Mit+17; DOR95], but it is not trivially evident that executable versions of these technologies are compatible with aspect-oriented modifications of their modelling language [CMM10; Cap+09]. As noted, low-level program transformation technologies are also unsuitable. Techniques for applying variations to models exist, but are unsuitable for simulation (and therefore cannot represent application based on temporal state) [SA13], produce individual models representing each possible instance of a variation [Pou+15], do not support the dynamic weaving of aspects for contingent behaviour [Mac+11], or attempt to represent the changes one would expect in the output of a simulation by executing an unmodified simulation and amending its output directly [SM14]. None of these techniques are suitable for representing a behavioural change which is applied contingent on state. Incidentally, these techniques for modelling behavioural variation also lack support for the alteration of a process definition, or changes "inside" a process definition, as discussed earlier, which also makes them unsuitable for simulated behavioural variation.

2.4.1 Research Questions

A tool which dynamically weaves definitions of behavioural variations for contingent application and which is capable of expressing changes within a join point rather than before or after it prompts a demonstration of the benefits that tool achieves. Contingent application of behavioural variation, and the ability to define changes to processes specifically, would fulfil the opportunity in marrying aspect orientation and modelling behavioural variation, but the benefit offered by such a tool is unclear. The introduction of oblivious modification to a model may break its representation of its real-world analogue, making such a model difficult to reason about. Added to this is the complexity of the tool's capacity to rewrite any join point's definition. Though aspect-oriented programming literature often lacks case studies demonstrating the benefits of the approach, it is particularly important to investigate whether such a tool could produce realistic models, and whether the expected benefits of aspect orientation as applied to the model hold in practice. Further, the research should demonstrate that the resulting aspectual modules are re-usable when modelling other systems: if so, aspects would represent modifications to models which cut across different model codebases.

This is investigated in the research presented in the following chapters. This literature review motivates the following research questions, which those chapters address:

> **RQ1** *Is an aspect orientation tool which is appropriate for use in simulation & modelling feasible to design?*
>
> **RQ2** *Can models of systems more accurately reflect their subjects through advice-based improvements?*
>
> **RQ3** *Can advice be used to accurately introduce behaviours or parameters into a model which were not originally present in it?*
>
> **RQ4** *Can advice be used as a portable module, such that aspect-oriented improvements to one model can be woven into another without loss of performance?*

To address these, new tooling is produced in chapter 4 which offers new types of aspects which weave their advice within their targets, rather than before or after them; a system is presented in chapter 5 around which a model can be constructed to investigate aspect-oriented simulation & modelling. That model — and the aspects applied to it — is described in chapter 6; this chapter addresses the first research question while laying the foundation for experiments which address the latter three. Those experiments are discussed in chapter 7. Before discussing these responses to the literature, related work which predates the contributions of later chapters is explained. Chapter 3 Distinguishes prior research from that contributed in this thesis.

Chapter 3

Prior Work

Lots of past tense here — where possible & appropriate should probably be present.

An implementation of the main tool developed and used in this thesis, "PyDySoFu", predates this thesis. As context for the contributions in this thesis, this chapter will describe the state of the project before the presented research was undertaken. Motivations for the tool's original development are described in section 3.1, which are followed by its design and implementation in section 3.2, and that of related tooling for experiments and simulations in section 3.3. The chapter concludes with a description of the research undertaken using these tools in section 3.4, as some results in the representation of behavioural variance using aspect orientation were found using these tools which predate this thesis and offer important background for the research undertaken in it.

3.1 Motivations in originally implementing PyDySoFu

The original incarnation of PyDySoFu was developed with different motivations than those outlined in chapter 2. It is therefore important to elucidate the context in which it was designed and developed.

PyDySoFu was originally developed for the representation of behavioural variance in sociotechnical systems, and was first produced as a proof-of-concept; the core contribution was one of tooling. It was developed for use in Python codebases, by virtue of the language's widespread use and its flexibility in its modelling of data and process.

The original version of the tool was to be applied to models of behaviour in socio-technical systems, where individual actions were represented as functions. Actions which could be decomposed further into more granular actions were to be defined as functions with calls to invoke their more granular counterpart functions. Invocations of low-level behaviours would implement some change to an environment in the model which its modelled behaviour would be expected to incur. Invocations of high-level behaviours, containing the invocations of lower-level behaviours they compose in the model, would therefore apply the combined effect of the collected behaviours they represent. A benefit of this approach to modelling behaviour was that high-level behaviours could implement the "flow" of a behaviour. For example, a behaviour which would be modelled in a flowchart as having some loop could be modelled analogously in the method described through use of primitive control flow operators in Python, such as `for` and `while` loops.

Another benefit of this approach is that the behaviours modelled have a predictable structure which is amenable to metaprogramming. A low-level behaviour's affect could be changed by changing the function definition; more structural changes could be made by altering the flow of less granular behaviours. A simple high-level behaviour containing a series of function invocations (modelling an ordered list of steps in the socio-technical system) can be represented as a literal list of function calls. The contents of such a list is trivially modifiable. Removing an item from a list or truncating it at a certain length, for example, are both achievable in a trivial manner using high-level languages such as Python. Notably, many behaviours can be conceived of which could be represented as high-level behaviours but would not be amenable to a simple list of more granular behaviours, such as a behaviour with a looping quality.

With a mechanism to rewrite either an implementation of a behaviour or a collection of behaviours (in the less granular functions mentioned), modelling in such a fashion could therefore lend itself to semantically simple metaprogramming that could represent real-world variations in behaviour. However, for reasons discussed in section 2.3, the use of metaprogramming to represent realistic behavioural variations in socio-technical simulations should be able to take advantage of system state. Many real-world behaviours are contingent on environmental state. Real-world actors in socio-technical systems might become tired after lots of work, or proportionally to time of day within a simulation. Therefore, the metaprogramming as described should be performed during runtime, for which no suitable candidate was available. PyDySoFu was developed to fulfil this requirement, so that behavioural variance in socio-technical simulation could be modelled as described and subsequently studied.

3.2 Original PyDySoFu Implementation

To disambiguate the improvements made to the original PyDySoFu implementation in the tooling contributions of this thesis — and to explain the fundamental concepts involved in both implementations' approaches to aspect orientation and the application of behavioural variance — the implementation of the original PyDySoFu tool, which predates the work presented in this thesis, is described here.

3.2.1 Weaving mechanism

The original PyDySoFu implementation made use of a weaving mechanism which could be categorised as "total weaving" in the parlance of Chitchyan and Sommerville [CS04]: hooks to apply advice are woven into every possible join point. The library was implemented in Python, which offers a flexible object model PyDySoFu is able to take advantage of in order to weave its hooks.[1]

Python's object model has three key properties which the original implementation of PyDySoFu takes advantage of:

① Everything in Python is an object, including types, functions, and classes

② Objects are, in essence, implemented as a dictionary[2] with string keys. All attributes of an object — such as a method on an instance of a class — are values in this dictionary, and their identifiers are the string keys of the dictionary. In essence, `someObject.val` is notionally equivalent to `someObject.__dict__['val']`, though the subtleties of this mechanism will be explained

[1] The weaving mechanism of PyDySoFu was eventually factored out into another library, Asp [WS23]; as PyDySoFu was mostly used before this refactoring, particularly in case studies (see section 3.4), the weaving mechanism will here be described within the context of PyDySoFu specifically.

[2] Python's name for what other languages might call a `map` or `hashmap`.

later.

③ Operations on objects are handled by "magic methods", which are specifically named methods on objects which Python calls for fundamental behaviour in the language. For example, `objA == objB` is interpreted by Python as `objA.__eq__(objB)`. Other built-in behaviours in Python have similar reserved method names which represent the implementation of some behaviour. These methods can be overridden, or specified by a programmer if they wish.

PyDySoFu weaves aspect hooks into classes by taking advantage of these three properties of Python. At a high level, PyDySoFu operates by replacing the `__getattribute__()` method of a class object with a custom one. `__getattribute__()` is the magic method responsible for performing lookups in an object's underlying dictionary. The replacement `__getattribute__()` also looks up attributes in the relevant object's underlying dictionary, but it also searches for advice to be applied when performing these lookups, and applies any advice it finds. The replacement `__getattribute__()` method represents the aspect hooks woven by PyDySoFu.

Hooks are applied to a class by way of an invocation to a function, `fuzz_clazz`, which takes a class as a parameter and weaves aspect hooks into that class [WS18c; WS23]. `fuzz_clazz` replaces the `__getattribute__()` method of the class with a new function object which it constructs. The replacement function object injects aspect hooks to callable attributes, thereby discovering aspects, applying them, and invoking an attribute which is the target of advice within a wrapper of the attribute itself.

The replacement `__getattribute__()` method makes a call to the class' original `__getattribute__()` method to retrieve an attribute when required. If this attribute is not a function or method, it is returned by the woven `__getattribute__()` function and the program affected class behaves as if it was never altered. However: if an attribute to be retrieved is a method or function, a new function is constructed and returned. This function looks up and applies advice to the attribute originally retrieved by the program, and contains a reference to the original aspect to enable its execution. This wrapping function is therefore the core of the weaving process: a replaced `__getattribute__()` method injects the aspect hooks which constitute the aspect-orientation framework. Advice to be run before, around, and after a join point were implemented as invocations of advice functions before a target was executed for "before" advice, after a target was executed for "after" advice, and with an around function being run with a reference to the advice's to invoke at its own discretion.

3.2.2 Applying Process Mutations

The development of PyDySoFu was intended to support simulation & modelling research by providing a way of applying program modifications at runtime. It sought to fulfil the needs of a program which determined that a (potentially non-deterministic) change to the behaviour it modelled was required, and enable the program to apply that change mid-process in an aspect-oriented fashion. Aspect orientation libraries typically support advice woven before, around, or after a join point; modifying the join point itself essentially allows changes inside its definition, introducing a fourth type of weaving. PyDySoFu achieves this through a special type of "before"-style aspect, which it calls a "fuzzer".

Fuzzers implement transformations on abstract syntax trees. They are implemented as functions which receive a list of AST objects representing the definition of a function which satisfies a fuzzing advice's join point, and return another list of AST objects, which replace the target function's definition. Any transformation resulting in a valid AST is permitted. A code snippet demonstrating the implementation of this process is shown in fig. 3.1.

As can be seen in fig. 3.1, fuzzing aspects are implemented by way of "prelude" advice (PyDySoFu nomenclature for advice run *before* a target invocation, inspired by early work on TheatreAG, which is discussed in section 3.3.2). Prior to the target being run, the fuzzer replaces its underlying code object with one produced by compiling a fuzzing aspect's returned AST objects. Replacing the underlying code object with a modified one allows the fuzzer to define arbitrary modifications to the definition of the function, thereby achieving runtime metaprogramming.

3.2.3 Limitations

The original implementation of PyDySoFu was intended to demonstrate the feasibility of runtime metaprogramming and its potential in simulation and modelling. However, its design presents

```python
class FuzzingAspect(IdentityAspect):

    def __init__(self, fuzzing_advice):
        self.fuzzing_advice = fuzzing_advice

    def prelude(self, attribute, context, *args, **kwargs):
        self.apply_fuzzing(attribute, context)

    def apply_fuzzing(self, attribute, context):
        # Ensure that advice key is unbound method for instance methods.
        if inspect.ismethod(attribute):
            reference_function = attribute.im_func
            advice_key = getattr(attribute.im_class, attribute.func_name)
        else:
            reference_function = attribute
            advice_key = reference_function

        fuzzer = self.fuzzing_advice.get(advice_key, identity)
        fuzz_function(reference_function, fuzzer, context)


def fuzz_function(reference_function, fuzzer=identity, context=None):
    reference_syntax_tree = get_reference_syntax_tree(reference_function)

    fuzzed_syntax_tree = copy.deepcopy(reference_syntax_tree)
    workflow_transformer = WorkflowTransformer(fuzzer=fuzzer, context=context)
    workflow_transformer.visit(fuzzed_syntax_tree)

    # Compile the newly mutated function into a module, extract the mutated function code object and replace the
    # reference function's code object for this call.
    compiled_module = compile(fuzzed_syntax_tree, inspect.getsourcefile(reference_function), 'exec')

    reference_function.func_code = compiled_module.co_consts[0]
```

Figure 3.1: A code snippet from the original PyDySoFu implementation, implementing `Fuzzing` aspects and applying fuzzing to a function definition.

limitations.

There is an overhead involved in running the wrapped function for every invocation of `__getattribute__()`, and also in running aspect hooks for all possible join points, even when those hooks are not targets of advice at a given moment. When an attribute is the target of advice, aspects are discovered and applied. However, aspects are discovered by lookup within the scope of the function creating a replacement `__getattribute__()` method. This design requires multiple instances of advice weaving to create multiple replacement `__getattribute__()` calls, all of which are invoked on any attribute lookup. A single target of advice which has multiple pieces of advice woven therefore incurs a performance penalty for every piece of advice applied, which must be incurred when any attribute is looked up on the target's class. If the join point defining the target may apply to many classes, each class must incur the same penalty, even if none of their attributes are targets of advice in practice. The original library was developed to demonstrate the feasibility of the idea underlying PyDySoFu (runtime metaprogramming), but the weaving mechanism implemented left room for improvement. A robust implementation with attention paid to reducing this overhead is introduced in chapter 4.

Similarly, an additional overhead is incurred by a lack of caching of the modifications fuzzers make to function definitions. One can envisage a need for runtime metaprogramming which produces different function definitions at different times: an example could be modelling different degrees of degraded modes introduced to an actor's behaviour in safety-critical systems research [JS07]. One can also envisage no such need: an example could be minor temporary modifications otherwise permanently made in program maintenance, such as to constants within a function definition, to the format of a function's return value, or adding control flow which exits a function early on a termination condition. The requirement is a product of the tool's use case in different scenarios. In scenarios where the same modification is to be made every invocation, a fuzzer need only be run once; optimisations enabling the caching of fuzzing aspects' effects would provide better performance in use cases where such a feature is appropriate.

Other aspect orientation frameworks offer support for other types of advice. Handi-wrap and AspectJ both support features related to the processing of exceptions thrown by a program [BH02; Kic+01] and these features have inspired work into improved exception handling in object-oriented systems [MD11]. However, this version of PyDySoFu offers no direct support for exception handling.

Opportunities to support the feature were therefore available for future revisions of the library to capitalise on; such a revision is also presented in chapter 4.

A final limitation is that the weaving technique this early of PyDySoFu used is incompatible with Python3, as replacing `__getattribute__()` is not possible in Python's newer version. It was determined that a tool which was of practical use to the simulation and modelling community should be produced which would remain useful to future researchers making modern models; as the existing weaving technique lacked performance, an opportunity presented itself for a complete redesign. The resulting new design is presented in chapter 4 which makes use of a new weaving technique.

3.3 Additional Simulation Machinery

Other related projects developed tooling for sociotechnical simulation & modelling. Fuzzi-Moss was a project collecting a library of standardised behavioural modifications for use in sociotechnical simulation & modelling; Theatre_AG was a project offering a model of time against which actors within sociotechnical simulations & models could act. While these projects ultimately were not used in producing the contributions of this thesis, they are outlined here as relevant to the original PyDySoFu project as they were originally developed as a suite of simulation & modelling tools to be employed together.

3.3.1 Fuzzi-Moss

Fuzzi-Moss[3] was a library of standard behavioural variations written as fuzzers to be applied by PyDySoFu [SW16]. It was primarily created for use in a model of the impact of inconsistency in teams' executions of software engineering methodology, which is discussed further in section 3.4.

Fuzzi-Moss contained utilities and fuzzers such as:

- Several probability mass functions representing chance of unexpected behaviour given a length of time and an actor's…:

[3] A backronym for "Fuzzing Models of sociotechnical simulations"

```python
def missed_target(random, pmf=default_distracted_pmf(2)):
    """
    Creates a fuzzer that causes a workflow containing a while loop to be
         prematurely terminated before the condition
    in the reference function is satisfied.  The binary probability
       distribution for continuing work is a function of
    the duration of the workflow, as measured by the supplied turn based
       clock.
    :param random: a random value source.
    :param pmf: a function that accepts a duration and returns a
       probability threshold for an
    actor to be distracted from a target.
    :return: the insufficient effort fuzz function.
    """

    def _insufficient_effort(steps, context):

        break_insertion = \
            'if not self.is_distracted() : break'

        context.is_distracted = IsDistracted(context.actor.clock, random,
             pmf)

        fuzzer = \
            recurse_into_nested_steps(
                fuzzer=filter_steps(
                    fuzz_filter=include_control_structures(target={ast.
                       While}),
                    fuzzer=recurse_into_nested_steps(
                        target_structures={ast.While},
                        fuzzer=insert_steps(0, break_insertion),
                        min_depth=1
                    )
                ),
                min_depth=1
            )

        return fuzzer(steps, context)

    return _insufficient_effort
```

  - conscientiousness, a lack of which would increase chance of behavioural adaptation due to lack of effort;
  - concentration, a lack of which would increase chance of behavioural adaptation due to distraction

- A `missed_target` fuzzer, which terminated a `while` loop early if activated via a probability mass function of an actor's propensity for negligence. As an example of a Fuzzi-Moss fuzzer, this is shown in ??.

Consider moving some code snippets like those of Fuzzi-Moss, which aren't necessarily that important in the grand scope of the thesis, to an appendix.

- An `incomplete_procedure` fuzzer, which truncated the steps[4] taken by an actor if activated via a probability mass function representing an actor's propensity for distraction

Plans were also made for fuzzers representing an actor "becoming muddled"[5] and making mistakes in decision-making, but neither have been completed at time of writing. A discussion around the revival of this project in the context of the PyDySoFu rewrite presented in chapter 4 is given in ??.

3.3.2 Theatre_Ag

Theatre_Ag ("Theatre") is a project defining a model of time against which actors in sociotechnical models & simulations can act [Sto23]. In the project's overview, it describes itself as...:

"T"heatre_Ag is a workflow oriented agent based simulation environment. Theatre_Ag is designed to enable experimenters to specify readable workflows directly as collections of related methods organised into Plain Old Python Classes that are executed by the agents in the simulation. All other simulation machinery (critically task duration and clock synchronization) is handled internally by the simulation environment.

The central metaphor underlying Theatre's model of timing is theatrical: actors in a simulation or model are members of a "cast" (a collection of actors) who enact a "workflow" (simulation steps) in a "scene" (domain model within which the actors interact). Central to the library is its clock: tasks are given durations, and a clock which synchronises all agents' position in time ticks to complete different tasks. The theatrical model theatre introduces is the context for PyDySoFu's nomenclature for its

[4] Represented by lines of code
[5] A behavioural variation caused by confusion.

types of advice: "prelude" advice happens before a task and "encore" advice is invoked afterward, as a prelude and encore would be in a literal theatre.

Theatre has been used as the environment in models of TCP/IP, algorithmic trading, the spread of disease [OLe20], and the impact of behavioural variation in software engineering methodologies as described in section 3.4.

3.4 Example Studies using PyDySoFu for Behavioural Simulation

The viability of encoding behavioural variations as aspects using PyDySoFu has been demonstrated in earlier studies [WS18a; OLe20]. The study modelled software engineers working to different methodologies of software engineering: waterfall, in which requirements are gathered, software is developed to meet requirements, quality assurance steps are undertaken, and the resulting software is delivered to customers; and TDD, where the development of tests for quality assurance precedes the development of features. The study sought to investigate whether, when software engineers were working suboptimally, there was a difference in the rate of bugs introduced to a program developed under each methodology.

Maybe I should include something on Aran's masters dissertation too, rather than just citing it?

The study began with a "naive" model of software engineers following each paradigm, devleoped in Python using PyDySoFu, Theatre, and Fuzzi-Moss. Engineers would produce "chunks" of code, which could contain bugs. In quality assurance, engineers were modelled as attempting to identify bugs in different areas of the codebase, fixing them if they were discovered. Developers could commit chunks of code toward features identified through requirement engineering, which were eventually completed, but could potentially contain undiscovered bugs within chunks of code.

This model was then augmented aspectually using PyDySoFu. Distraction was represented through the truncation of functions representing workflow steps. Developers were modelled with different levels of distraction, affecting a probability mass function (PMF) which would activate when a developer was modelled as being distracted in a given moment. If the PMF activated, the workflow step invoked at that moment was truncated using PyDySoFu. The model showed that developers following the TDD methodology could successfully complete a larger number of features on average than those following

Add a citation for TDD vs agile! Citations are missing from the copy of the CAiSE paper I can find, but I'd like to use the same one.

waterfall, concurring with the prevailing consensus on the two methodologies.

In replicating the community understanding of the model, the paper demonstrated the feasibility of aspectually augmenting modelled behaviour: the simulation took a naive model with no capacity for analysing errors, and introduced new features of the model supporting an avenue of investigation otherwise impossible with the methodologies represented by the naive model. That the resulting simulation matched the expectations existing within the community gave confidence that the tool could be used to build realistic simulations where some features of a model could be separated from its core codebase.

3.5 Discussion

The existing case study employing PyDySoFu demonstrated that realistic model features could be separated from their core codebase, and gave credence to PyDySoFu's use as a tool for aspectually augmenting models with behavioural variance. It also left many research questions unanswered and tooling flaws unsatisfied, however:

Does this want to be a list? Consider reworking to a paragraph.

- Aspects were believed to be "realistic" as they represented the expected outcomes of the simulation. However, no real-world data was used to corroborate the claim, and it was unclear that aspectually augmented behaviour could capture the variations present in real-world human behaviour.
- These aspects also demonstrated variations which were identifiable in the emergent properties of a system (for example, mean time to failure of a software system under development, or successfully completed features). The variations applied to individual developers might have poorly modelled individual behaviour, but produced accurate emergent properties of the system individual developers acted within.
- As discussed in section 3.2.3, PyDySoFu's implementation at the time was a proof-of-concept which, while successfully demonstrating the potential of aspect-oriented runtime metaprogramming, was also inefficient, feature-incomplete, and lacked compatibility with modern software engineering tooling.

→ Models of distraction were adopted from the common library provided by Fuzzi-Moss. However, this model was not applied to other codebases. It remains unclear that Fuzzi-Moss' model of distraction is broadly applicable in other projects: different models of distraction might be required by different researchers. Further, a model of distraction which realistically represents the behaviour of an individual (rather than the emergent properties of the system that individual acts within) might not apply to other systems the individual acts within. Briefly put, The portability of aspectually modelled behavioural simulation has not been investigated, and literature within the aspect orientation community lacks evidence to support a belief in their portability [Prz10; CSS04; Ste06].

This left opportunities to improve both the tooling offered for aspect-oriented runtime metaprogramming, and the evidence supporting its use to encode behavioural variations in sociotechnical systems. Improvements to the tooling follow in chapter 4; later chapters propose — and discuss the implementation of — real-world systems which are suitable for modelling using PyDySoFu in chapter 5, and a study of those systems using aspectually augmented models in chapter 6 and chapter 7.

Chapter 4

Rewriting PyDySoFu

Check the most up-to-date pdsf implementation — is it actually on that backup drive?

The work undertaken in this thesis required in improved implementation of old tooling. When previously used, PyDySoFu was a proof of concept which could feasibly produce scientific simulations, but was implemented in a manner which was not optimised for speed (making it a burden for large simulations), lacked granularity in the application of its aspect hooks (hooks could only be applied to entire classes), and most importantly, did not work with Python3 (Python2 support officially ended during this PhD).

This chapter briefly outlines the new implementation of PyDySoFu, discusses improvements made to design and performance, and explains some contributions made to the design of aspect orientation frameworks which addresses some core issues raised with the paradigm . Consider adding references to the sections through this PDSF chapter, depending on how beefy it becomes…

Rewrite chapter outline after the structure of the chapter is known to be OK.

4.1 Requirements for Change

As time wore on with PyDySoFu's original implementation, it became increasingly clear that a rewrite was required . PyDySoFu grew out of an undergraduate project, and accrued technical debt as a result of being written under extreme time constraints with little experience. On revisiting, and on

Too informal

sloppily put

reflecting on other aspect orientation frameworks reviewed in section 2.1 and the use previously found for PyDySoFu [WS18a; WS18b], it was clear that there were improvements which could be made in the process of rewriting the tool:

- Before this body of work, PyDySoFu made use of techniques for applying aspect hooks which did not translate to the changes Python 3 made to its object model. In particular, Python 3 changed its underlying object model, using a read-only wrapper class that made the replacement of `__getattribute__` impossible via the previous route.

- PyDySoFu's original implementation made no serious accomodations for efficiency. It could be seen as the "total weaving" described by Chitchyan and Sommerville in [CS04], and it was not possible to provide additional options to ensure that aspects could be as efficiently woven as possible at runtime given a particular use-case .

  Could be less sloppy

- The original PyDySoFu implementation wove onto a *class*, meaning that even properties of the class which were not considered join points were still affected by the weaving, even if in a minor way . Because `__getattribute__` retrieves all attributes including special builtin attributes and non-callable attributes, these are also returned via the modified implementation of `__getattribute__`, incurring an overhead, albeit small, for all attribute resolutions instead of a desired subset.

  Could be less sloppy

- The original PyDySoFu implementation made no accommodations for scenarios where fuzzing of source code was applied in a "static" manner. That is to say, where a deterministic modification to source is woven as advice, instead of dynamically modifying source code, the same modification would still be made every time the target attribute was executed, unless caching of results was specifically managed by the aspect applying the change. No optimisations were made pertaining to this, but compilation and abstract syntax tree editing have the potential to be PyDySoFu's most expensive operations.

- Unlike other aspect orientation frameworks such as AspectJ [Kic+01], join points could not be specified by pattern. Instead, each individual join point must be supplied as a Python object. This means that, while the target attributes are still oblivious to the advice applied to them, the application of that advice could not be written obliviously.

As a large number of requirements were left unfulfilled by the original implementation of PyDySoFu, a new implementation satisfying them was deemed necessary.

4.2 Python3-compatible Weaving Techniques

Replacing `__getattribute__` on the class of a targeted method was no longer viable in Python 3. A replacement method therefore had to be found. For clarity: replacing `__getattribute__` allowed for hooks to be woven (at runtime) into likely future targets for advice. These hooks would then discover and manage the execution of advice around each target. Because advice can be run before and around a target, and dynamic weaving implies that advice could be supplied or removed at any time, we look to intercept the calling of any target, and manage advice immediately before execution. So, the task at hand is to find a method of attaching additional work to the calling of any potential target, before that target is executed. We refer to code woven around a target which manages applied advice as *aspect hooks*.

4.2.1 Abandoned techniques

Rather than "monkey-patching"[1] a new version of `__getattribute__` with hooks for weaving aspects, the rewritten method could be patched to the object itself at a deeper level than used in the original PyDySoFu implementation. This would make use of Python's `ctypes` api to patch the underlying object. Similar work has been done in the python community in a project called ForbiddenFruit [con21]. Efforts were made to add the required functionality to ForbiddenFruit — patching `__getattribute__` directly on the object, or "cursing" it in ForbiddenFruit jargon — but this was abandoned as the underlying mechanism is particularly unsafe, Python API changes could render the work unusable in future versions easily, and the implementation would only work with particular implementations of Python (for `ctypes` to exist, the Python implementation must be written in C). Community patches existed for cursing `__getattribute__` which did not work,

[1] Monkey-patching is the practice of making on-the-fly changes to object behaviours / definitions by taking advantage of language properties such as flexible object structures. Common examples of these structures are objects literally being maps from string attribute / method names to the associated underlying value, as in Python and Javascript. Monkey-patching makes use of these simple structures by replacing values such as the function object mapped to by the original function's name in the dictionary, effectively changing its behaviour. This is the method by which PyDySoFu originally replaced `_getattribute__` on a class object.

and attempts proved challenging, indicating that this would also be complicated to maintain over time. There are also efficiency concerns with this technique depending on its use: weaving advice around a function would mean monkey-patching the built-in class of functions, which would incur an overhead from running aspect hooks on every invocation of every function.

Another approach involved making use of existing Python functionality for interrupting method calls. As PyDySoFu wraps method calls at execution time, what is required is to add functionality to the beginning and end of the execution of a method. Python has built-in functionality for implementing debuggers, profilers, and similar development tools, which provides exactly this functionality, as debuggers must be able to — at any point during execution marked as a breakpoint — pause a running program and inspect call stacks, the values of variables, and so on. As a result, the method `settrace()` allows a developer to specify a hook providing additional functionality to a program. Making use of this also has issues in our case. Most significantly, `settrace()` catches myriad events in the Python interpreter which PyDySoFu may not need to concern itself with, incurring significant overhead. In addition, use of the function overrides previous calls to it, meaning that any debuggers used by a user of PyDySoFu would be replaced with PyDySoFu's functionality, which was deemed untenable. However, it is worth noting that the technique could work in theory, and if future versions of Python allow for multiple trace handlers being managed by `settrace()`, this could provide an interesting approach when implementing future dynamic aspect orientation frameworks.

4.2.2 A viable technique: import hooks

A final available technique was to continue to monkey-patch hooks to discover and weave aspects, via an alternative method which did not make use of `__getattribute__`. This approach would change the use of PyDySoFu slightly to make a compromise between performance and obliviousness of aspect application: when importing a module targeted for aspect weaving, methods which are potential weaving targets are invisibly monkey-patched with a wrapper method with a reference to the original[2] and hooks to detect and run dynamically supplied advice.

An important note for discussing the implementation of PyDySoFu is that almost all Python

[2] Necessary to run the originally targeted method.

functionality operates by use of its "magic methods". "Magic methods" are methods beginning and ending with two _ characters. The Python language documentation specifies sets of magic methods and their required function signatures which are used internally to implement functionality; for example, any object with the method `__eq__()` defined can be compared against using the == operator, and the `__eq__()` magic method is run to determine the outcome of the operator. Magic methods support more than operator overloading. For example, anything which defines `__len__()` and `__getitem__()` is treated as an immutable container, and adding `__setitem__()` and `__delitem__()` makes that container mutable. Any class defining `__call__()` is treated as a callable object (not unlike a function). More can be found in Python's documentation[VD09], although more focused guides exist in the Python community [con16]. For the purposes of implementing import-based aspect hook weaving, magic methods have the affect of making the language ideal, particularly as an environment to implement dynamic aspect orientation. Python's functionality for importing modules is managed by `builtins.__import__`, which receives module names as strings and handles package resolution. By monkey-patching the import system, modules can be modified during the process of importing. As this technique allows for control over where aspect hooks are applied, PyDySoFu can target only function and method objects to apply aspect hooks to, avoiding the overhead its previous iteration introduced when applying hooks to all attribute lookups including non-callables, such as variables or `Class` objects.

Monkey-patching `builtins.__import__` is as simple as replacing the function object with a new one, which has the effect of changing the behaviour of Python's `import` keyword: because all Python functionality relies on magic methods implicitly, its behaviour can be altered in this way. However, our intent is not necessarily to manipulate *all* modules, but a subset of imports specified by a modeller as suitable for manipulation. If all invocations of `import` wove hooks into modules, including those made in the process of importing packages, an unnecessary overhead would be introduced when invoking any callable in any module. Therefore, it is important to have a mechanism to enable and disable the weaving of aspect hooks for a given `import` statement (effectively, to enable and disable PyDySoFu's modified import logic).

This can be achieved through another use of magic methods in a manner which also makes clear to a modeller exactly where aspect hooks are being applied: making use of Python's `with` keyword. A

```python
with AspectHooks():
    import mymodule
```

Figure 4.1: Example of importing a module, mymodule, using PyDySoFu's import hook design.

more technical discussion on the process of weaving, and the nature of the hooks applied, follows in section 4.3.

4.3 Import Hooks

4.3.1 Implementing import hooks

We are interested in manipulating `builtins.__import__` only when imports are made which introduce modules containing prospective join points; a developer might import many modules, which do not all require aspect hooks to be introduced. We enable this new import behaviour with a syntax of the form shown in section 4.3.1, which weaves aspect hooks into all functions and (non-builtin) class methods within the `mymodule` module object added to the local namespace of the importing stack[3]. Less formally: importing `with AspectHooks()` applies aspect hooks to all potential targets of advice in the `mymodule` package. The behaviour of Python's `with` keyword is defined by more magic methods: any object with `__enter__()` and `__exit__()` defined can be used here, where `__enter__()` is run at the beginning of the enclosed block, and `__exit__()` when leaving the block.

PyDySoFu caches the original `builtins.__import__` object in an instance of the class, and replaces it with `AspectHooks.__import__`, in its `__enter__()` method. This is reversed by replacing `builtins.__import__()` with the cached object in its `__exit__()` function. `__enter__()` and `__exit__()` are the magic methods corresponding to entering and exiting Python's `with` blocks, meaning that this technique modifies Python's importing only when imports are made within a block such as that in section 4.3.1. The resulting implementation for weaving aspect hooks is uncomplicated, as can be seen in fig. 4.2.

[3]Python's use of the stack namespace in its importing system means that careless re-importing a module can lead to multiple copies of it in different function stacks, meaning that the same name resolution (such as resolving a class by its name in a module) might, after applying aspect hooks in PyDySoFu, change the behaviour of procedures depending on where they are called. Scenarios where this might arise are deemed unlikely enough that the risk of this design decision becoming troublesome are considered negligible. Still, it would be remiss not to make note of the fact.

```python
class AspectHooks:
    def __enter__(self, *args, **kwargs):
        self.old_import = __import__
        import builtins
        builtins.__import__ = self.__import__

    def __import__(self, *args, **kwargs):
        # ...replacement import logic for performing weaving...

    def __exit__(self, *args, **kwargs):
        builtins.__import__ = self.old_import
```

Figure 4.2: Magic methods used to enable the `with` keyword usage for PyDySoFu

4.3.2 Strengths and weaknesses of import hooks

As a technique for weaving aspect hooks, this new method provides multiple benefits. Application of aspect hooks is straightforward from the perspective of a modeller using PyDySoFu, whose code clearly applies aspect hooks and does so in a legible way for future maintainers. Import hooks' explicit application of hooks to modules means that places where aspect hooks might be applied equally explicit, and thereby implements an interpretation of aspect-oriented programming's principle of "obliviousness" moderated by clarity. While join points are oblivious to potential aspect application, callers of join points can expect to be more aware, and signs for codebase maintainers are left directly within source code. Aspect hooks can be applied to specific modules or every module depending on the use of the supplied `with` statement, allowing for total weaving or actual hook weaving [CS04] depending on their preferences. Further, performance is optimised in comparison to the previous implementation of PyDySoFu, as hooks are weave-able at a more granular level (on the level of procedures such as functions and methods, rather than all attributes of a class).

There are also caveats of this approach that are necessary to address. As aspect hooks are woven in the new implementation of PyDySoFu via Python's import functionality, any procedure not imported from a module cannot have aspect hooks attached. Consider adding local namespace weaving to pdsf3: should be easy to implement as a cheeky little monkey-patch... However, as aspect orientation is primarily concerned with a separation-of-concerns approach to software architecture, targets are expected to exist in other modules, and we do not consider this to be a significant limitation.

A more significant limitation of the import hook approach is that the object with aspect hooks woven exists in the namespace of the module *importing* the join point. In other words, this method makes it impossible for a module to make use of aspect hooks that are woven in an unrelated piece of code. is this true??? verify, I think not, if imported normally. tested but didn't realise that pdsf won't fuzz functions with single-character IDs, so inconclusive. We therefore have a "semi-oblivious" property to our aspect orientation approach: targets of advice are unaware of any adaptations made, but *any code making use of those adaptations must be aware enough to at least apply aspect hooks.*[4]

In a manner of speaking, this can be considered to alleviate some concerns with aspect orientation as a paradigm. Aspect Orientation is criticised for making reasoning about programs more difficult [Prz10; CSS04; Ste06]. One cause of this is that aspects separate logic from where it is run; Constantinides, Skotiniotis, and Stoerzer 's comparison with the jokingly proposed `come from` statement [Cla73; CSS04] is a reminder that it can be effectively impossible to understand how a program will execute if the path of execution is not at least linear or clearly decipherable from source code. Aspect orientation as a paradigm violates this linearity by design. However, import hooks as implemented in fig. 4.2 allow for aspect-oriented code which can be interpreted in one of only two ways:

① Looking at the original implementation of a procedure, its intended execution is clear. A programmer can make use of this directly and it is guaranteed to behave as expected.

② Any program making use of a procedure imported from a module will see, when the procedure is imported, whether it has had aspect hooks applied. In this case its behaviour is unknown — falling prey to the design flaws discussed in the aspect orientation literature **??????** — but this unpredictability is highlighted to the programmer.

As a result, while import hooks are somewhat limited in that they are applied specifically to imported code and break the traditional AOP concept of obliviousness in at least a weak manner, these two facts combine to arguably fix a latent issue in the design of the aspect oriented paradigm. The original PyDySoFu implementation was able to modify any procedure in a more traditional, oblivious manner. While this new implementation is clearly more limited as a result, we consider these limitations an overall benefit to the design of the tool, and a contribution to aspect orientation framework design.

[4] Note that once aspect hooks are applied, advice can still be supplied from anywhere in the codebase.

4.3.3 Weaving process

Weaving in PyDySoFu's updated implementation takes place via monkey-patching of aspect hooks, as described in **??**. Aspect hooks replace executable targets within a module at the moment the module is imported. When the target is invoked, the wrapping aspect hook is executed in lieu of the original target object. The wrapping aspect contains the target function within its closure, allowing it to execute the original target; however, it was also created by the `AspectHooks` class, and so has reference to aspects registered against it.

The process of weaving with this implementation of PyDySoFu is therefore as follows:

① An aspect is imported using `AspectHooks`, as shown in section 4.3.1.

② An aspect is registered against `AspectHooks` as shown in fig. 4.3, by providing a regular expression matching the ID of any join points to apply an aspect to and the aspect to apply. Specifically relevant to aspect registration:

- Methods permitting aspect application are:
  ① `AspectHooks.add_prelude(rule, aspect)`, which registers advice to be run before a target is invoked
  ② `AspectHooks.add_encore(rule, aspect)`, which registers advice to be run after a target is invoked
  ③ `AspectHooks.add_around(rule, aspect)`, which registers advice to wrap an aspect invocation, effectively providing the functionality of `prelude` and `encore` advice with a single aspect
  ④ `AspectHooks.add_error_handler(rule, aspect)`, which registers advice to catch and process exceptions raised by a join point
  ⑤ `AspectHooks.add_fuzzer(rule, aspect)`, which registers advice to modify a target before it is invoked, effectively providing aspect application within a join point, and permits arbitrary ast-level modifications of the target
- Each method described above also accepts an optional `urgency` integer parameter, enabling optional additional features of PyDySoFu which are discussed in section 4.4.3.

- Invoking the above aspect hook registration methods compiles and caches the regular expression provided for efficiency when identifying matching join points in later invocations of aspect hooks.
- Each method outlined above returns a callback which de-registers the aspect provided, to facilitate the ephemeral application of aspects. This could be useful in experimental codebases where aspects represent behavioural deviations, and a researcher looked to run simulations with different deviations applied to compare datasets, for example.

A bit clunky — take the last sentence out in favour of notes in discussion or rewrite.

③ Invocations of executable objects within the module imported in ① are replaced with aspect hook wrappers of themselves; an invocation of an executable object within that module therefore triggers PyDySoFu's aspect hooks, which seek aspects registered against the ID of the target they wrap. Aspects registered against a join point identifying rule which matches the ID of the wrapped target can therefore be identified, and executed as appropriate.

All aspects applied when registering advice are callable objects; however, they have different function signatures. The arguments expected by each type of advice is:

prelude advice takes `target, *args, **kwargs`

encore advice takes `target, original_return_val, *args, **kwargs`

around advice takes `next_around, target, *args, **kwargs`

error_handler advice takes `target, handled_exception, *args, **kwargs`

fuzzer advice takes `ast_steps_from_target, *args, **kwargs`

…where `target` is the target an aspect is applied to; `original_return_value` is the value returned by a target after it was run; `next_around` is what an `around`-style advice runs when it has finished its pre-target component, and intends to run the target itself; `ast_steps_from_target` is a list of AST objects representing the definition of the target a fuzzing aspect is applied to; and `*args, **kwargs` is a Python idiom collecting arbitrary positional and keyword arguments passed to a function but not specified within its function signature, which are here employed to collect arguments

```python
def log_invocations(target, *args, **kwargs):
    '''
    An aspect to print invocations of a target.
    '''
    print("Invoking " + target.__name__)
    target(*args, **kwargs)
    print("Invocation of " + target.__name__ + " finished.")

from pdsf import AspectHooks
with AspectHooks():
    from some_module import some_func

AspectHooks.add_around("some_func", log_invocations)
```

Figure 4.3: Registering an aspect against AspectHooks

passed to a target, and can be used by any aspect to inspect the expected behaviour of the target (or to pass into the target, if the aspect invokes it).

The execution of `prelude`, `encore`, and `error_handler` aspects work predictably: `prelude` aspects are executed before the wrapped target, `encore` aspects after the wrapped target, and `error_handler` aspects within a `try` block encompassing the execution of all advice. Notably, the return values of `around` aspects replace those of their target, and `encore` aspects replace that of the target if they return any value which is not `None`. By comparison, the implementations of `around` and `fuzzer` aspect weavers are non-trivial.

As with all aspects, `around`-style aspects are given the target being invoked alongside its supplied positional and keyword arguments. However, these aspects are also given the next `around`-style aspect woven against the target to run. Each aspect should be provided its `next_around` argument. However: with a series of many `around`-style aspects to apply, and each calling the next in the series, a given aspect would need to know the `next_around` parameter for the aspect it calls; the second could call a third, meaning it would need to know the `next_around` parameter for its own successor, which could only come from the first in the series. The third could call a fourth, requiring another `next_around` parameter, which the second would have to pass to the third, and the first to the second. The inelegance of this naive solution to the problem seems confusing and inelegant for modellers to interact with or understand.

Decide whether the "-style" prefix I sometimes use should stay or go when copyediting. Or: when do I use it? Be consistent.

However, two alternatives present themselves. One is to use Python's generator pattern, which can be simply described as a callable which returns successive items from a series on each invocation.

```python
from functools import partial, reduce
final_around = lambda target, *args, **kwargs: target(*args, **kwargs)
nest_around_call = lambda nested_around, next_around: partial(next_around
    , nested_around)
target_with_around = reduce(nest_around_call, around_advice, final_around
    )
```

Figure 4.4: Simplified example of weaving `around` aspects in PyDySoFu

An alternative approach is to supply each `around`-style aspect with its first argument (the next in the series) before the aspects are invoked, unburdening individual aspects with responsibility to retrieve and apply the next aspect successfully. This approach is employed by PyDySoFu. The first argument of every `around`-style aspect is provided by constructing a partial function. Partial functions are a feature of many languages which allow for some arguments of a function to be provided, but for the function not to be invoked; instead, a new function is returned, with any remaining arguments left to be provided on invocation. With the first argument provided, the resulting partial function has the signature `target, *args, **kwargs`, where all values are known and trivially supplied by any `around`-style aspect.

The problem to solve is therefore the construction of the partially applied aspects in such a way that each refers to the next correctly. To solve this, each function is provided to its precursor; iterating through all `around`-style aspects in this manner results in a single aspect which contains a reference to the second, the second a reference to the third, and so on until the final registered aspect. The final aspect has no successor it can reference however; to overcome this, a function which does nothing but execute the original target is provided, with the same signature as the partially-applied aspects. The final function terminates the chain, and is responsible for executing the target itself; every other aspect is simply responsible for the execution of the next `around`-style aspect in its chain.[5] In this way, `around`-style aspects can be woven in a manner which simplifies a developer's interaction with the aspects. A simplified snippet of the source code of PyDySoFu which implements this process is shown in fig. 4.4 to illustrate.

The final kind of advice to apply are `fuzzer`-style aspects, which PyDySoFu contributes to aspect orientation framework design. These aspects make modifications to the definition of the target they are applied to, allowing advice to be woven within a target. Arbitrary transformations of the target also

[5] Note that this is technically optional; if it determined it was necessary, any `around`-style aspect could break the chain by calling the target with its arguments directly.

```python
if fuzzers is not None and fuzzers != []:
    code = dedent(inspect.getsource(t))
    target_ast = ast.parse(code)
    funcbody_steps = target_ast.body[0].body
    for fuzzer in fuzzers:
        non_inline_changed_steps = fuzzer(funcbody_steps, *args, **kwargs
            )

        if non_inline_changed_steps:
            funcbody_steps = non_inline_changed_steps

    target_ast.body[0].body = funcbody_steps
    compiled_fuzzed_target = compile(target_ast, "<ast>", "exec")
    if not isinstance(t, FunctionType):
        t.__func__.__code__ =  compiled_fuzzed_target.co_consts[0]
    else:
        t.__code__ = compiled_fuzzed_target.co_consts[0]
```

Figure 4.5: A code snippet representing the process of modifying a target of an aspect application using a `fuzzer`-style aspect. `t` refers to the target on which modifiations are applied.

allow removal or change of any part of the target definition. This is achieved by acquiring the original source of the function to modify using Python's built-in reflection library, `inspect`. The source can be parsed into an abstract syntax tree using Python's built-in library, `ast`, which contains a list of AST objects representing a target's original definition. This list of AST steps is then passed to a `fuzzer` aspect as an argument, which returns a new list of AST steps if any changes are to be made. The new steps are compiled and the resulting Python `code` object containing a modified function definition is monkey-patched into the original target, effectively changing its definition. The original code object is cached during aspect hook weaving, and is used to replace the modified `code` object after target execution to avoid unexpected behaviours in future invocations of the target. A simplified example of the process of acquiring, modifiying, compiling, and monkey-patching a target's underlying definition is found in fig. 4.5 .

4.4 Optimisations

Additional features were implemented in the presented incarnation of PyDySoFu which improve its usability in the research software engineering context, and optimise performance where necessary: deep hook weaving, non-dynamic weaving, and aspect priority support.

```
from pdsf import AspectHooks
AspectHooks.deep_apply = True
```

Figure 4.6: Code snippet enabling deep hook weaving

```
from pdsf import AspectHooks
AspectHooks.treat_rules_as_dynamic = True
```

Figure 4.7: Code snippet enabling dynamic weaving

4.4.1 Deep Hook Weaving

Under ordianry circumstances, PyDySoFu weaves aspect hooks into an imported module, but not modules imported *by* that module. Deep hook weaving enables weaving of aspect hooks into all potential join points by applying hooks to functions and methods imported by the original module, but also any recursive imports. A code snippet enabling the feature is presented in fig. 4.6.

Deep hook weaving introduces performance overhead due to additional checks for aspects which were woven dynamically. However, it is plausible that the modules within which desired join points reside are not imported directly by a developer, but are available to them indirectly through another package they make use of. It is also plausible that a developer may look to instrument the entirety of — and investigate details of — a call stack without attaching a debugger to a process. In both examples, PyDySoFu's default behaviour is insufficient. Deep weaving is therefore provided to support these use cases.

4.4.2 Non-Dynamic Weaving

PyDySoFu's runtime weaving of aspects allows for aspects to be applied at any time. Aspects can be removed and re-applied during program execution. However, programs may be written with the intention of applying aspects once — as program modifications to be introduced without direct manipulation of a codebase — so that, once defined, the set of applied aspects would remain static. In this scenario, the dynamic weaving of PyDySoFu introduces unnecessary overhead. Weaving aspects dynamically means that every invocation of a prospective join point requires searching for aspects which *presently* apply to it.

```
from pdsf import AspectHooks
AspectHooks.manage_ordering = True
```

Figure 4.8: Code snippet enabling priority ordering of aspect application

To avoid this overhead in scenarios where it is unrequired, PyDySoFu can cache the aspects applied when any callable wrapped by an aspect hook is invoked for the first time. On its first execution in this mode, the aspect hook stores the set of aspects it matched to the invoked target, and future invocations retrieve the set of aspects to apply from the cache. This avoids expensive regular expression matches, which fail in all cases but those where an invoked target is to be augmented by the application of an individual aspect the regular expression is paired with.

Add metrics here on the performance implications; they're pretty significant, I actually added them because the regex matches slowed my own code down to a crawl. Our experiments make use of non-dynamic weaving for this reason.

For performance reasons, the default mode of PyDySoFu is to run with non-dynamic weaving. Dynamic weaving functionality is enabled by toggling the `treat_rules_as_dynamic` flag on the `AspectHook` class, similarly to deep hook weaving (see section 4.4.1). A code snippet demonstrating this is shown in fig. 4.7.

4.4.3 Priority Ordering of Aspect Application

As dynamic weaving allows for non-deterministic application of aspects, it may be that the order in which aspects are intended to be applied is not the aspect in which they are woven. To support these use cases, aspects may be sorted by priority during the lookup of aspects to apply when a join point is invoked.

As mentioned in section 4.3.3, when aspects are registered against the `AspectHooks` class, an optional `urgency` parameter is available. The parameter takes an integer representation of priority of aspect application, with higher numbers representing more urgent application. Aspects with no urgency applied default to `urgency=0`. High-urgency aspects are applied before low-urgency aspects. This feature is disabled by default, but can be enabled using the code snippet shown in fig. 4.8.

4.5 Discussion

contributions to what?

Is this list complete?

The new implementation of PyDySoFu makes a few contributions, particularly in comparison to the previous version:

- Introduces a new technique of weaving aspect hooks on import, improving its design over a typical aspect orientation framework by making use of Python's `with` keyword when weaving hooks to trade a degree of obliviousness for clarity
- Aspect hooks can be applied with more precision than the previous implementation of PyDySoFu, meaning:
  - Users of the framework can better delineate between total and actual hook weaving
  - Unnecessary overheads from checking dynamically applied aspects at each join point are reduced.

PyDySoFu's current incarnation also provides opportunities for improvements and for future work. Our intended use case for aspect orientation for simulation & modelling is in scientific codebases specifically; direct integration with the scientific package ecosystem (which is vibrant in Python's community) should be made. A good initial project would be integration of aspect application in sciunit tests [Tha+17]. A discussion on potential use cases of PyDySoFu together with existing research software engineering technologies is provided in section 8.7.

Chapter 5

RPGLite: A Mobile Game for Collecting Data

RPGLite is a game designed to mimic existing popular games, while being structured to permit exploring all game states via formal methods; Kavanagh and Miller have produced PRISM models which can be model-checked to identify optimal play strategies in all game states [KM20] . Some experiments were conducted around RPGLite to answer the question: "Over time, do players converge on an ideal strategy of play?" Fix formatting and correct wording of William's hypothesis. Review note from Jess: this is *very* important (and, yes! I don't believe i have my RQs well-defined anywhere in the thesis yet, which is concerning.) Insert my own research question here

Real-world play of the game invites many research questions. For example, an alternative question to answer would be, "what strategy of play do players typically adopt?" or, relatedly, "do all players adopt the same strategies?" [1] Kavanagh and Miller's work can identify the *cost* of an action [KM20; Kav21], allowing for richer datasets to be produced and more in-depth analysis of real-world play to be conducted. One possibility invited by these datasets is that it may permit modelling of real-world players' interactions with the game, and therefore offers an opportunity to investigate the effectiveness of aspect-oriented simulation and modelling. Referring back to TODO INSERT RESEARCH QUESTION HERE , it would be feasible to build a model of RPGLite play which did not satisfactorily model real-world data, inviting aspectual augmentation of the model in an attempt to improve its performance, and thereby investigating the research question proposed.

[1] These are not of research interest for the purposes of this thesis, but examples of interesting questions to ask of a game where "correct" and "incorrect" actions can be categorised.

About what, and why? Consider changing title. add a sentence in first paragraph describing why this fits in with the rest of the dissertation, and why it is here.

Suggest new sentence

PRISM undefined

"game states" not welld efined here, though I do define it later on

Lift screenshots in this chapter from game-on pa-

However, it would not be possible to perform analyses of player behaviour without real-world player data. Datasets collected empirically would be required to compare against synthetically generated datasets, to ascertain their similarity and thereby compare different methods of producing synthetic datasets. As RPGLite is a real-world sociotechnical system which already yields a formal analysis of play, and datasets from this system can be used to support multiple avenues of research in different disciplines,[2] producing a dataset of real-world RPGLite could also support future work in a variety of fields as a non-commercial dataset for research in fields including at least game design and gameplay research [Kav21], formal methods [KM20], sociotechnical simulation research, and research software engineering tooling demonstration.

J: rework passive voice

To that end, a collaboration was undertaken with Kavanagh to develop and release a mobile implementation of RPGLite which would collect player data for research purposes. The dataset produced would enable Kavanagh to demonstrate the utility of their model checking in an empirical scenario [Kav21], and for the purposes of this thesis, it would also enable the analysis of models representing player behaviour, by supporting the comparison of these models against the collected data.

J: this para's easier to understand than previous ones in this chapter. When reworking, consider how the others could be rewritten in a style closer to this? She also notes it might belong further up in the chap.

J: avoid passive voice

J: She suggests moving this earlier, too

The development and release of RPGLite as a mobile game offers an opportunity to make use of aspect orientation in a new context: a model of naive RPGLite play can be produced which represents random play, and aspects could be written which augment the naive model representing hypothesised player behaviour. If the aspectually augmented models generate data which correlates with empirically sourced data more closely than that generated by the naive model (with random play, and no aspectual augmentation), we can dismiss naive play as "realistic", and understand the aspectually augmented behaviour as "more realistic". Many aspects can be written representing different styles of play, which might be adopted by different players, a concrete benefit of aspect orientation in modelling & simulation.

The modelling & simulation research opportunities RPGLite presents are realised in chapter 6 and chapter 7; this chapter focuses only on the design and development of RPGLite as a mobile game, and the dataset collected from real-world play and released [Wal+21]. It starts with an overview of RPGLite itself in section 5.1, and progresses to a discussion of the game's implementation in section 5.2 before

[2] Later contributions in this thesis are supported by datasets produced by RPGLite, as were the contributions of Kavanagh's PhD thesis [Kav21].

concluding with a discussion around the empirical data collected using the game in section 5.3.

5.1 An Overview of RPGLite

Needs copy-editing

RPGLite is a simple two-player game played in turns. Each player selects characters independent of the other, with each character having a unique set of abilities and properties, which are generally health, chance of success on attack, and damage dealt on a successful attack. The abilities of some characters necessitate additional properties. Each player selects an "alive" character (one with health greater than 0) to perform their action against a chosen "alive" target (or occasionally targets). A successful attack — randomly determined by chance of successful attack for the selected attacking character — results in that character's unique ability being inflicted on their target[s]. A random player is chosen to take a first move, players may always skip their turn as a valid action, and players continue to take alternating turns until a victor is left with the only "alive" characters.

Eight characters are available for selection, with the following abilities :

J: compress the description list. Maybe make it bulleted? I wonder whether it can just be a para or something.

Knight Deals damage to an opponent character on a successful hit.

Archer Deals damage to two opponent characters on a successful hit.

Wizard Deals damage to an opponent character on a successful hit, disabling (or *"stunning"*) them for the duration of the opponent's next turn.

Healer Deals damage to an opponent character on a successful hit, and heals themselves or, optionally, the other player character instead (assuming that character is still alive).

Barbarian Deals damage to an opponent character on a successful hit, dealing additional damage if their health is low when attacking.

J: whose? the barbarian's? Clarify.

Rogue Deals damage to an opponent character on a successful hit, dealing additional damage if the target's health is low when attacked.

Monk Deals damage to an opponent character on a successful hit, and immediately takes another turn, until their attack is unsuccessful.

Gunner Deals damage to an opponent regardless of success, dealing additional damage on a successful hit.

Specific details of each character — their health, chance to hit, and damage on hit as well as character-specific details (such as the threshold for additional Barbarian or Rogue damage, for instance) — are defined as a *"configuration"* of RPGLite. Different configurations change the game's *"balance"*, a term referring to the relative strengths of different characters or character pairs. For example, if a configuration leaves many characters with initial health values close to a Barbarian's threshold for additional damage, then they become a very powerful character due to their ability to inflict additional damage. If the Monk's chance to hit is high, the repeated turns it offers can be very advantageous. Character skills can work in concert with each other: choosing a Barbarian and Healer such that the barbarian can be kept at low health for additional damage, but the healer can be used to keep them alive, may be an effective strategy depending on the game's configuration. Kavanagh et al. found that model-checking a configuration of the game could discover the relative strengths of characters and character pairs when played optimally [Kav+19].

RPGLite's design has two objectives it must meet. First, that it is interesting to players, which requires that it is approachable and complex enough not to be immediately solvable. This is necessary for real-world data collection, and to demonstrate a design representative of something that could conceivably be a real-world game with an active playerbase. Second, RPGLite's design must be sufficiently simple for model-checking. Model checking is a necessary requirement of design because of our need to identify optimal moves: analysis of player behaviour rests on our understanding of how close to "ideal" players are, and whether players approach ideal strategies over time. This is the crux of the work found in formal methods research on RPGLite [Kav21; KM20], which relies on a reduced state space in order to calculate optimal moves and character pairings.

5.1.1 RPGLite's Design Implications

Jess noted this wasn't formally described. I describe it later, but clearly it's confus-

The state space of RPGLite makes it unusually well-suited to analysis through formal methods. Because of this, datasets produced through simulation of RPGLite can be compared against two other datasets: one of real-world play, and another of what can be mathematically shown to be "correct" player behaviour.

To demonstrate this state space, note that RPGLite games can have their states described by a set of values: the healths of characters on each team, plus a stunned character. With eight characters each having their own maximum health value, which we denote w, x, y, z[3], two players, and an indicator of which character is stunned.[4]. The entropy of a game's state is therefore $log_2(w \times x \times y \times z \times 3 \times 2)$, where the multiplications by 3 and 2 represent the stunnedness indicator for the current player (either character, or neither), and the player whose turn it is to play, respectively. The maximum health for a character is 10 hit points, which makes the maximum entropy of a game state $log_2(10^4 \times 3 \times 2\}) = log_2(60000) \approx 16.87 bits$. Each player picks 2 of 10 characters, and can choose to attack either opponent character with either of their own, or skip their turn, for a maximum of 6 possible actions. We can therefore see that the total entropy of the entire RPGLite game is no greater than $log_2(10^4 \times 6 \times 2 \times \binom{10}{2} \times 6 \approx 24.36 bits$.[5]

J: so is that the state, or just a description of a subset? (It's a complete and uniquely identifying state, but I don't make this clear enough)

Iterating through these states allows us to map the entire state space of RPGLite. As the state space the game defines is relatively small[6] it is feasible to analyse every possible game state. Note that the figure of $\approx 24.36 bits$ includes movements *between* states as well as the states themselves, meaning that it is feasible to map the potential progressions through all possible games of RPGLite using formal methods. Further, this allows us to understand the chances of a player winning given transitions between different states, for example by representing moves in the game as transitions in a decision diagram where nodes are the game's state. We can calculate exactly how "good" a move is, by comparing chances of success when making a given move in a given state against the chances of success when making the objectively optimal move.

J: define earlier? Also: why is entropy important here?

J: does the order matter?

J: Summarising her notes but 1. Footnote should be integrated into body and 2. Entropy's a confusing concept, wasn't what she was expecting here for defining a state space. Either make the concept extremely clear or, better, avoid the concept of

[3]Maximum health values are dependent on RPGLite's configuration.

[4]Note that stunnedness is valid for exactly one character for one turn, meaning that only one character may be stunned at any time, and the status effect immediately resets, meaning there are only three possible states for stunnedness: either character belonging to the player taking a turn, or neither.

[5]The actual figure is smaller:

- Players may choose two characters with special abilities that prevent them from attacking both opponent characters at once (this accounts for 9 out of 10 characters), giving 5 possible actions in a turn rather than 6.
- Players are likely to judge different strategies to be more effective at different times due to their assessment of others' strategies: certain decisions might be poor in general but successful in certain circumstances. If a player determines that a certain strategy is "popular", they may assume their opponent will play in the popular manner and adapt their own strategies accordingly [How71; Kok+21]. RPGLite is designed to have slightly "unbalanced" parameters of different characters such as health, attack damage, or potency of special abilities, meaning long-term players are expected to learn effective playstyles and adjust accordingly.

Players' behaviour is therefore less uniform than this calculation would imply, but the calculation provides a higher bound on the entropy of the game.

[6]For comparison, mapping valid positions in chess takes about 136 bits [Nie77], and this figure does not account for valid *moves* within the game, which our calculation for RPGLite does

J: first section isn't clear, she notes she didn't understand. In second section she circles "known", asking, "known by whom?"

In this way, RPGLite's design allows it to be understood formally, yet it also draws on common game design elements and is sufficiently "interesting" to generate data from a real-world playerbase.[7] This yields some useful properties for the purposes of aspect-oriented simulation :

1. Simulated moves can be selected naively, i.e. at random, but can also be made perfectly according to the known-correct move in a given game state, or made with some calculated "cost" as to the chance of winning.

2. Real-world players' behaviour can also be analysed according to the same metrics: for example, moves made by players can be analysed to understand bias, whether players learned to play "better" moves over time, or whether they selected known-strong characters more frequently than those who can be formally shown to have a relatively low chance of winning games.

3. As the actions taken when playing RPGLite are consistent — such as deciding the target character of an attack, or a character to use in an attack — random play can be simulated as a "naive" play style, which can be compared against real-world players. Where player behaviour does not correlate to naive play, [8] the biases of players may be represented as aspects which are applied only to specific actors within the simulation.

Footnote suggests correlation is defined in chap 6; include a more specific cref.

4. Should aspects be suitable as a manner of accurately representing biased play, aspects offer a separation of concerns within the simulation: any nuance found within the playstyle of specific real-world players would be replicated and applied not to the model itself, but to specific simulated players. Playstyles might also be mixed with the application of multiple aspects.

Whether aspect orientation is suitable for the realistic simulation of RPGLite gameplay is the topic of the remainder of this thesis. However, the design of RPGLite allows for a controlled system where a clear notion of "good" and "naive" behaviours can be defined, the system is closed insofar as all interactions within the system are known and all game elements are precicely understood, and all interactions take place between experiment participants for data collection purposes, allowing for a large dataset to be collected without information being removed due to players not consenting to their data being collected and disseminated for science. In short, RPGLite's design constitutes a system where all aspects are well-understood, no interference is anticipated from system components which are unknown or outside

[7] Demonstrated through data collected from several thousand completed games, which can be found at [KWM20].
[8] The concept of play correlation is introduced in chapter 6.

of experimental control, and lots of data can be collected for analysis. Therefore, the data generated by gameplay is suitable for comparison against datasets outputted through aspect-oriented simulation. This can be used to assess whether simulated players with aspect-affected behaviour accurately reflect the playerbase, and so can help to assess whether aspect orientation is suitable for realistic simulation.

5.2 Implementation of RPGLite

J: compress

Knowing ideal play is useful. However, to understand how real-world players would interact with RPGLite, empirical data needed to be collected. To produce this, a mobile online multiplayer version of the game was developed for data collection purposes. Play constituted engagement with an experiment for data collection, and after several months, a database logging player behaviour offered a dataset which could be used to simulate real-world player behaviour.

This section describes details of RPGLite's implementation as a data collection tool. Some lessons learned after reflection on the implementation process were documented for the benefit of others' avoiding our errors[Wal+21] .

Should a summary of the paper be a section here, or maybe an appendix? Keen not to plagarise, but it's also maybe worth including. William and I both spent a lot of time getting that paper out. I presented it at GameOn2020. He's included

5.2.1 Consent to participation

Players gave consent to participate in a scientific study as a part of creating an account to play RPGLite. Players were required to explicitly scroll through an information sheet and consent agreement and agree to both to make an account. Copies of both were made available to download for player reference in both the game and RPGLite's website[9] . Players were instructed to contact the involved researchers to withdraw participation, and were instructed that they could do so at any time. Email addresses to contact were noted in both the app and the website.

The study, information sheet, consent form, and data collected were reviewed by the University of Glasgow Science & Engineering ethics committee before the game was publicly released.

[9] RPGLite's website is available at https://rpglite.app/, where copies of the consent form and information sheet are linked.

5.2.2 Mobile app

As a mobile game, RPGLite's user-facing component was an application, distributed through the Google Play Store on Android and the Apple App Store on iOS. This was developed in Unity, a framework for developing games in C# which can be distributed to almost any platform. Most assets were developed in GIMP, with character designs contributed by a commissioned artist online. Unity allowed for a "WYSIWYG" or what-you-see-is-what-you-get interface builder, with event handlers defined in C# code which would "hook" into events signalled by interface element interactions. User-facing components of the game were largely produced by Kavanagh, the collaborator on this project and original RPGLite designer. Therefore, in an attempt not to take credit for this component of the work involved in collecting the RPGLite dataset, curious readers are referred to Kavanagh's notes on the development process for full details [Kav21].

J: rest of this para should be made shorter.

Beta testing required user engagement. Apps were deployed to Android and iOS devices of colleagues, who played a series of games to check that game logic was sufficiently robust and graphic design sufficiently adequate for public distribution of the game. In a manner inspired by design science methodology [JP14], beta tests comprised of iterations on the artefact of the game and its server-side component (discussed in section 5.2.3) which were distributed to beta testers. Informal feedback and enhancement requests were sought on each iteration until the game behaved correctly in all edge cases (no major bugs were reported) and a final design was settled upon (no major complaints about interaction or aesthetics were reported), at which point the game satisfied criteria for public distribution.

An example of the game's visual evolution is given in fig. 5.1, where colourful buttons replace a tabular, text-heavy interface. Another example is the evolution of the game's main screen, "Active Games", which loaded when players logged into the app and allowed users to see and interact with active games they were involved in. The visual identity and colour palette of this screen was refined iteratively, as can be seen in fig. 5.2. Other features which were developed following user feedback while beta testing include the implementation of a notification system and a leaderboard showing a player's experience relative to their peers; these are shown in fig. 5.3.

There are some good screenshots in the paper for RPGLite;

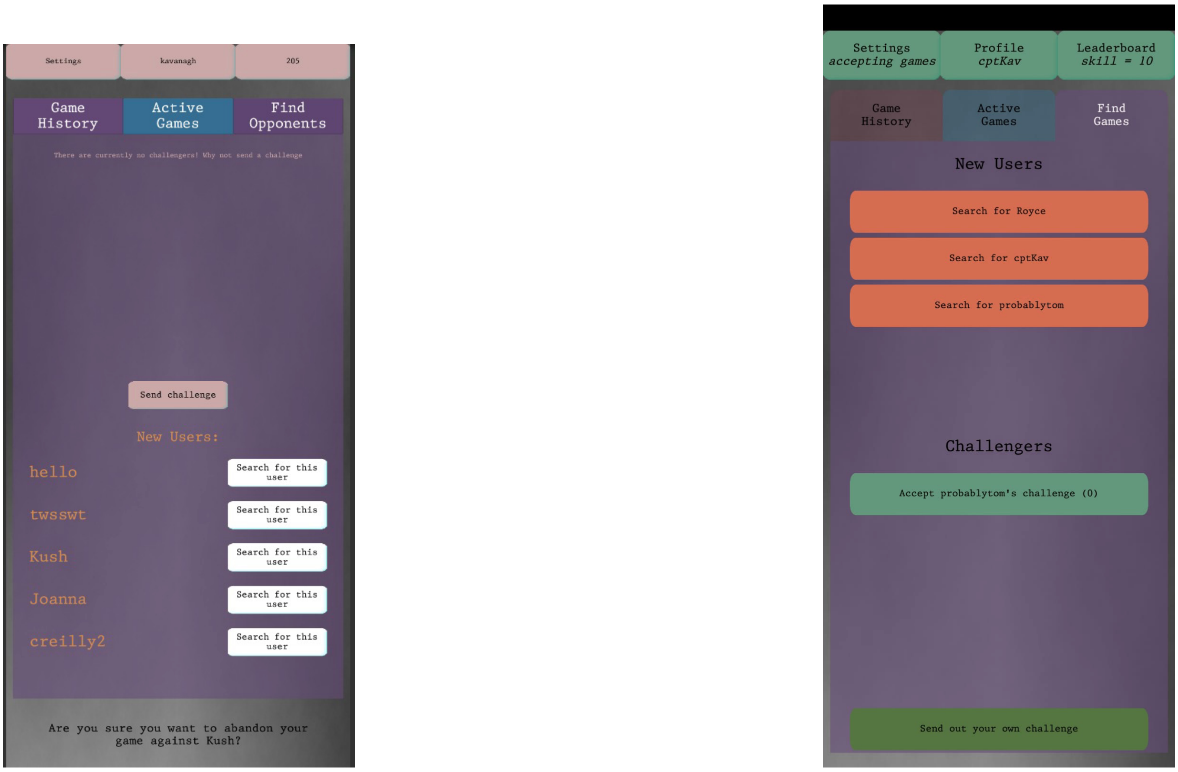

Figure 5.1: Evolution of the "Find users" matchmaking screen. Prototype left, final design right.

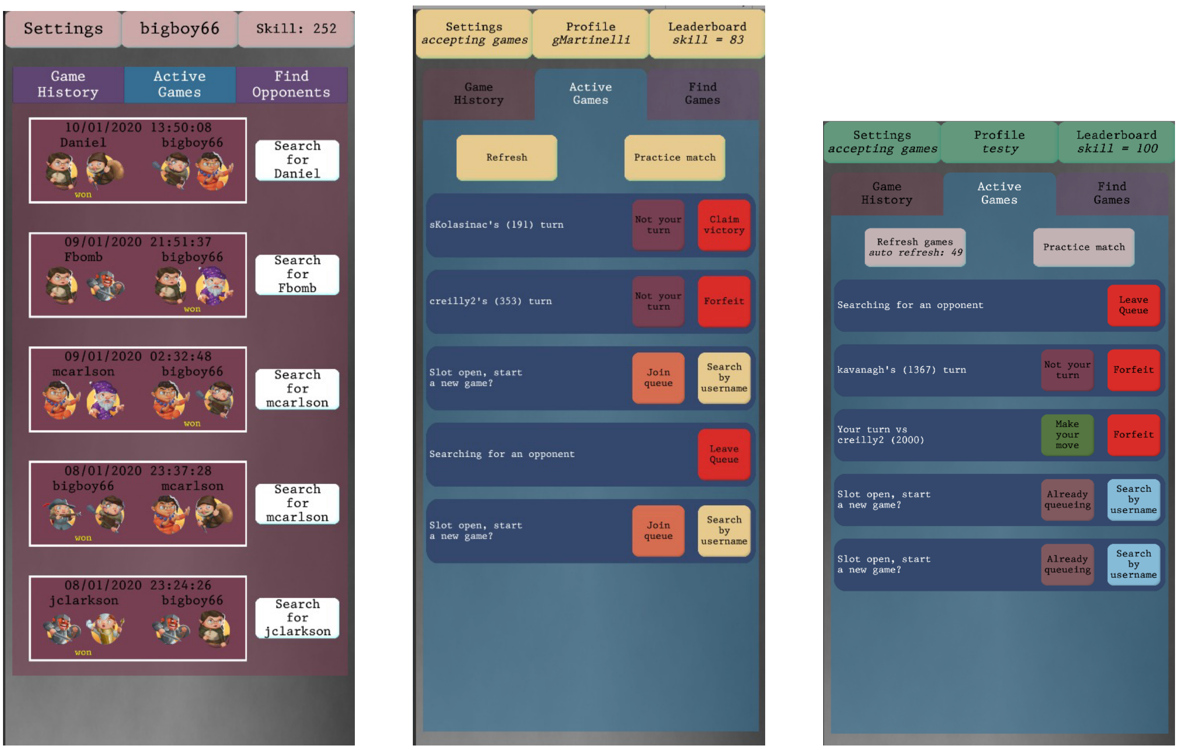

Figure 5.2: Evolution of the "Active Games" screen, which allowed RPGLite users to see and interact with games they were playing. An early prototype is shown to the left; a refinement through beta testing in the center; and the final version released to the public to the right, with an improved colour palette.

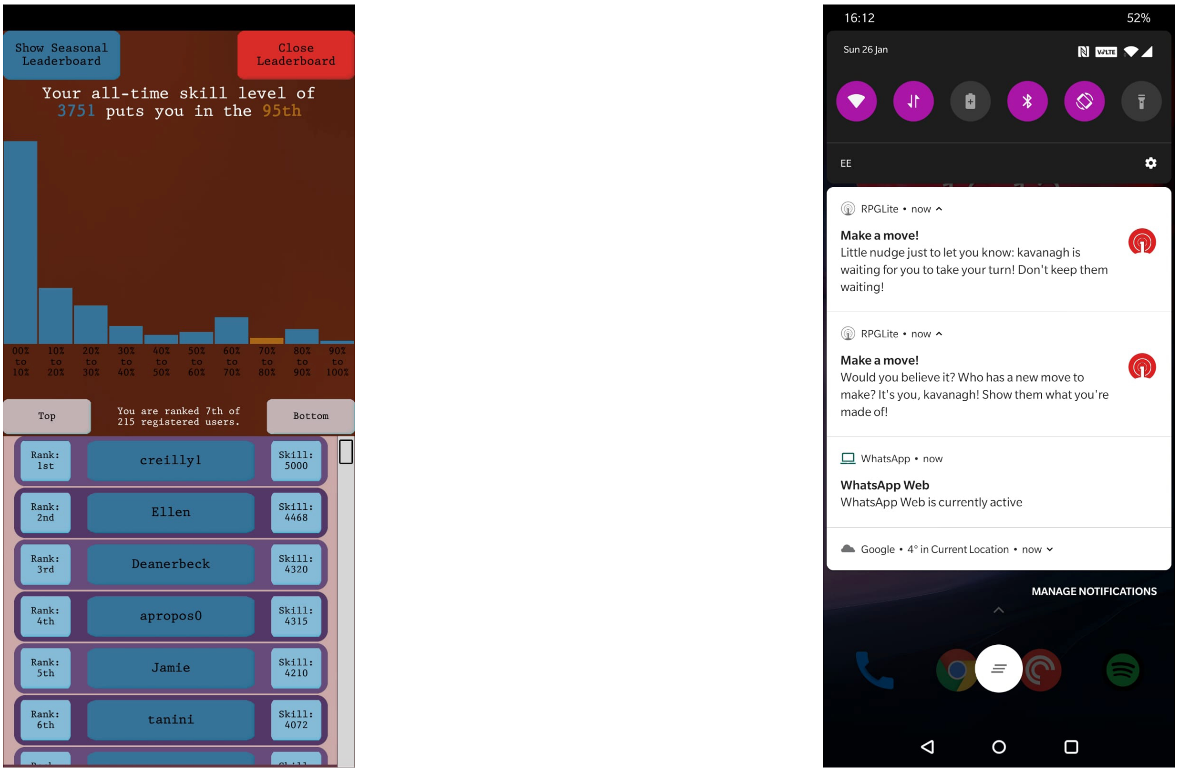

Figure 5.3: Features developed following requests from beta testers. A leaderboard of player engagement is shown to the left; examples of notifications when an opponent has acted are shown to the right.

5.2.3 API & Server-side Logic

J: how could the data not be empirical?

As the data collected ought to be empirical , RPGLite was developed as an online game. This required a server and API for a client to communicate with.

A REST API was developed with Python's *Flask* framework. Endpoints were created to support all in-game actions requiring centralised state (or a recording of that state), allowing for player search, matchmaking, player profile design, game history and statistics analysis, ranking calculations, login and password reset, synchronisation of access to sensitive information, and other in-game activities. The API also allowed moves to be made, and rejected erroneous game states or unauthorised input from any malicious actor, to ensure that the data collected, analysed, and published was not manipulated. The API also sent push notifications to an opponent's device when moves were made. This was a feature requested during beta testing, which was anecdotally observed to increase engagement with the game.

On each of these actions, data was collected about the action performed, and logged in a database. In addition, in-game activities which required no server-side input but were considered to have potential in later analysis would log datapoints through the API into the same database.

A MongoDB database instance was installed and managed on a University of Glasgow Computing Science virtual machine. The no-SQL nature of the database permitted flexible structuring of the data, and simplified analysis of the games' results after data collection was completed. The API was also hosted on the same virtual machine. A combination of port access rules applied via Nginx and hardening the security of the database itself through MongoDB's account features and controlled permissions within the server prevented unauthorised access to the database, ensuring that the data remained untampered.

Game state was interpreted client-side. The intention in this design choice was to un-burden a centralised service we were responsible for maintaining, moving functionality such as calculating game states after moves were created to clients on players' devices. This left the serverside logic mostly concerned with executing database transactions and ensuring the integrity of game states was uncompromised. Reflections on the suitability of this approach made after the game's launch have been published separately [**GameOn2020**].

5.3 Empirical Play and Data Collected

In total, players produced a dataset of 9,693 games. It also includes 1,105,065 datapoints generated by gameplay or player interaction with the client, such as players checking their position on a leaderboard, searching their game history, or rolling to hit as part of an attack. The data is drawn revnoteJ: present here, past elsewhere, be consistent with tenses directly from the MongoDB database used to run the game. With player consent on signup (see section 5.2.1) this data is published and available for all researchers to analyse in the format of a JSON object [KWM20].

J: suggests this is separate from games. Is that correct? (Yes; make this clearer.)

Completed games drawn from the MongoDB instance contain many fields. The breadth of datapoints logged by player interaction with the RPGLite app means there are an extremely broad range of datapoints available to curious researchers. A full list of available data is available in a text document within the dataset **??** ; in brief, some of particular interest include:

J: make more concise. (I made slight edits when adding this revnote, maybe the para can be

- The history of moves made, and the times those moves were made

- The players involved and the winning player of each game (by username as no personally identifying data was collected)
- The ELO scores of players in each game
- The characters chosen by each player
- The "score" of each player [10]

ELO undefined.

J: "score" is vague. I clarify in the footnote but it should be in the body here. Also, "score" is not of interest. We don't use it in the experiments, don't waste attention on it.

RPGLite was played through two "seasons": after an initial $3,000$ games played in a first configuration of RPGLite, a second configuration was released. This meant that, having learned a strategy to play with the initial configuration (a preference of characters, for example), changes were made which may invalidate what players had learned. The dataset therefore contains two conceptual subsets: data collected from a system with the same mechanics but slightly tweaked J: this chapter is actually unfinished. (I can't believe I missed this. OK. There's a little more to say on the dataset itself; set yourself up in particular for the parts of the data that are important in the experiments. Make note of the changes between S1 and S2; I think I'd meant to look this up, forgot, and thought the final section was complete, which is a serious oversight, but there's not too much left to write here.)

[10]RPGLite's mobile app presented users with a naive scoring mechanism used to rank users on a leaderboard which some users then used to identify other players of a similar notional skill.

Chapter 6

Designing and Implementing Aspect-Oriented Models

This chapter describes the design and implementation of experiments which investigate using advice to augment behaviour in a model. The experiments simulate RPGLite gameplay. Using these simulations, the experiments investigate the utility of aspect-oriented programming in a simulation & modelling context. Advice which uses data about players' gameplay [KWM20] is applied to a basic model of RPGLite play; the advice changes simulated players' behaviour to model individuals' interactions with the game. The behaviour modified by the advice is players' selection of character pairs. Character pairs selected by simulated players are compared against those from a player's gameplay data and their correlation is measured. If the datasets correlate, then players are successfully simulated. This approach is used to answer the last three research questions proposed in chapter 2:

> **RQ2** *Can models of systems more accurately reflect their subjects through advice-based improvements?*
>
> **RQ3** *Can advice be used to accurately introduce behaviours or parameters into a model which were not originally present in it?*
>
> **RQ4** *Can advice be used as a portable module, such that aspect-oriented improvements to one model can be woven into another without loss of performance?*

To answer these, two approaches to modifying player behaviour are developed: one alters the distribution from which character pairs are selected to reflect the distribution found in a player's gameplay data, and the other models the player learning to play RPGLite and developing preferences for certain character pairs in the process.

The experiments share a significant proportion of their design and implementation. As their common elements require a significant amount of explanation, a chapter outlining both the setup *and* results of these experiments would be extremely long, and their shared foundation means that they are most naturally explained together. For this reason, the design and implementation of *all* experiments is explained in this chapter, and their results follow in chapter 7.

The common aspects of design and implementation in this chapter are explored as follows. To give context as to how different pieces of the experiments' foundations are used, an overview of the models constructed and the experiments employing them is given in section 6.1. The model of RPGLite play which these experiments weave aspects into is described in section 6.2. The design of the more complex of the two models of behaviour change — a model of learning — is explained in section 6.3. Aspects implementing the learning model, other models of behavioural change, and additional apparatus required to conduct experiments are described in section 6.4. Having described details of the aspects used by experiments, details of the implementation of the experiments themselves are given in section 6.5. Following this, chapter 7 explains the design, implementation, results and evaluation of each experiment individually, as all context necessary to understand these designs and evaluations are given in the aforementioned sections.

6.1 Experiment Design

To answer the proposed research questions, two models are made which change simulated players' character pair selections: one which reflects the choices made by the player being simulated, and another which simulates that player learning over time. These are used to examine three things about aspect-oriented simulations & models: whether aspects can be woven to improve a model's accuracy; whether it is feasible to use aspects to introduce new behaviours and parameters in a model; and whether the same aspects can be successfully woven across different models.

The complete designs for the first, second, and third experiments are explained in section 7.2, section 7.3, and section 7.4 respectively as it is useful to have discussed the implementation of the RPGLite model, aspects, and statistical methodologies before the experimental design is explained. However, some context as to how those foundations are used is important when explaining them too. For this reason, a brief description of the behavioural modifications and the experiments which use them are given here, and more detailed descriptions are given in the following chapter.

6.1.1 Changes to Behaviour

Simulated players' behaviour is changed using aspects when they select characters. The rationale for altering character selection in particular is explained in section 6.3. The model they alter — described in section 6.2 — is "naive", meaning that it makes all decisions a player could make randomly rather than strategically or optimally. Two changes to character pair selection are created: the first selects characters with the same distribution as found the gameplay data of a player being simulated, called the "prior distribution" model; the second selects characters by modelling players learning which characters are most likely to win games and selecting character pairs according to what they learned, called the "learning model".

Prior Distribution Model The prior distribution model selects character pairs with a distribution calculated from players' gameplay data.[1] As the distribution is already known, simulated games can reproduce the distribution. This makes little change to the behaviour of simulated players, as no new activity is modelled on their part: no additional actions are taken and simulated players do not behave in a "new" way. However, the emergent properties of simulated play should reflect the dataset used by the prior distribution model. This is expected to have the effect of synthesising new datasets with the same emergent properties and requires little modification to the model. The implementation of the prior distribution model is given in section 6.4.3.

Learning Model The learning model selects character pairs by preferring pairs which previously won games. This model requires an additional model of confidence to support it (discussed in section 6.3.4) and special measures to be in place when generating datasets to ensure the space of

[1]The distribution is known ahead of the simulation being run, hence "prior".

possible character pairs is explored properly (discussed in section 6.5.4). As a result this explanation is superficial and is provided as necessary context for the introduction of experiments following in section 6.1.2. A thorough discussion of the learning model is given in section 6.3.

The learning model selects character pairs using a distribution defined by the pairs' number of observed wins. If a player has never seen a character pair winning a game, that pair is unlikely to be selected; if a player sees a pair winning an overwhelming number of games, the pair will be selected proportionally to its win rate. This model introduces new behaviours for simulated players, as their decisions are based on historical observations which are not present in the original model and emerge from a player's interactions with others as opposed to being defined when the model is first executed. New parameters are also introduced to control how players learn, including a parameter controlling their arrogance, a parameter scaling the number of games required for them to rely on their knowledge, and parameters controlling how bored they are and how likely they are to stop playing RPGLite. The introduction of new behaviours and parameters to the naive model allows the latter two research questions to be answered.

6.1.2 Brief Explanations of Experiments

Experiment #1: Improving a Model The first experiment answers the research question: *Can models of systems more accurately reflect their subjects through advice-based improvements?* Aspect-oriented programming can only be used to augment models if modifications to a model can be encoded in aspects, woven into a model, and become provably more accurate as a result. Similar research was conducted in prior work [WS18a], but this study represented changes to a model which were not verified as "accurate" or "realistic". Beyond producing a change that *looked* accurate, it did not simulate any real-world system and so could not be verified as rigorously as a model of RPGLite gameplay can be.

To verify that a model is changed in the intended way through the weaving of advice, three datasets of completed games are used for each player simulated. The first is that player's gameplay data collected from their interactions with the mobile game in RPGLite's first season, the second is produced by the naive model, and the third is produced by the naive model with the prior distribution model woven. The distribution of character pairs chosen by each model is compared with the player's distribution of character pairs chosen using a correlation metric explained in section 6.5.3. The random distribution of chosen pairs produced by the naive model is expected to correlate poorly with the player's distribution of choices, as the player is expected to be biased toward certain character pairs as they continue to play RPGLite. The prior distribution model should produce datasets with the same distribution as the player exhibited when interacting with the mobile game, so the dataset produced with the prior distribution model woven into RPGLite should correlate closely to the empirically sourced dataset. If this behaviour is seen, then the aspects woven into the model induced a change in simulated players' behaviours and so successfully augmented the model.

A more thorough description of this experiment is given in section 7.2 as the reader has been presented with the details of the experiment's foundation at that point.

Experiment #2: Extending Behaviours in a Model The first experiment investigates whether advice can alter a model and introduce changes to existing behaviours. Another experiment follows to investigate whether advice can introduce *new* behaviours to a model and new parameters which alter modelled behaviour. This aims to answer the research question: *Can advice be used to accurately introduce behaviours or parameters into a model which were not originally present in it?*

The learning model is woven into the naive model of RPGLite play to add new behaviours and parameters to the model. For each player, parameters are found using a technique explained in section 7.1.2 which simulate their learning most accurately. This technique determines reproducible results which show statistically significant correlation with the player's gameplay data from the RPGLite mobile game, extending the statistics used in the first experiment (explained in section 6.5.3). If the learning model can reproduce the play style of some real-world players, then the additional behaviour and parameters successfully model their play of RPGLite and the experiment is a success. To do this, datasets are generated by weaving the learning model parameterised in different ways. The correlation of these datasets is measured against subsets of a player's season 1 gameplay data. If the parameters can produce data which correlates with the player's gameplay data consistently, the player is accurately simulated by the learning model with these parameters and so the research question is answered.

Note that not all players are expected to be simulated accurately by the model of learning: players might have biases toward or against some characters, might play in cliques which limit their exploration of possible character pairs, or might learn differently to the way the model assumes such as exhibiting temporal discounting (see the work of Green and Myerson [GM96] for an example). As these experiments aim to investigate the feasibility of applying aspect-oriented programming in a simulation & modelling context, building a learning model which is universally applicable to all players is beyond the project's scope. The experiment only seeks to demonstrate that new behaviours *can* be accurately represented by advice, and that the technique can be used in future work to augment models in whatever manner a research team requires.

A more thorough description of this experiment is given in section 7.3 as the reader has been presented with the details of the experiment's foundation at that point.

Experiment #3: Behaviours as Cross-Cutting Concerns Having conducted the first two experiments, the utility of aspects as units of behaviour change are established. The results in section 7.2 and section 7.3 are positive. A final research question remains, which is: *Can advice be used as a portable module, such that aspect-oriented improvements to one model can be woven into another without loss of performance?*

It occurs to me that these *might* read as rhetorical questions when used in this way. I'm not sure so I'd love somebody to review the chapter and let me know how it comes across, so I can change this if necessary. I'm struggling to see the forest for the trees at this point.

To establish this, season 2 of RPGLite is used to represent a system which is subtly changed. RPGLite's seasons are defined by game configurations, which means that the behaviours of players do not change but the strengths and weaknesses of different characters do. To play strategically, players must learn which changes have affected their ordinary strategies and re-evaluate their preferred character pairs. The prior distribution model and learning model should work independently of the RPGLite season they are applied to, and so — given positive results are found for the first and second experiments — the models ought to work when woven into a model of the second season too.

Data is generated using the prior distribution model and learning model in the same manner as in earlier experiments, but applied to player's gameplay data from RPGLite's second season. Parameters which yielded statistically significant results for the learning model in season 1 are used in season 2, as these parameters ought to represent how an individual player learns. The learning model may also work to simulate season 2 players, but with different parameters; to investigate this, model parameters specific to season 2 are found which yield statistically significant datasets using the same technique as in the second experiment. If these aspectually-augmented models successfully simulate players then the experiment establishes the portability of aspects representing behavioural change. Correlation is measured in the same ways as in previous experiments to evaluate the models' accuracy in simulating players.

A more thorough description of this experiment is given in section 7.4 as the reader has been presented with the details of the experiment's foundation at that point.

6.2 Naive Model

A naive model of play was developed by separating each stage of the actions taken by players and separating them into individual procedures. The model was written as a workflow in Python, and state of workflow execution was separated into three components: the actor that a function invocation (or "step") represents activity from; a "context" in the parlance of languages like Golang, representing the state of a game being played; and the environment in which that game is played.[2] The naive model of RPGLite follows a simple workflow mimicking player interaction with the mobile game deployed. A graphical representation is provided in fig. 6.1. Figure 6.2 contains a diagram showing how the naive model is used to produce synthetic data by simulating repeated gameplay.

Move mermaid-based figures to .svg for fidelity

Two randomly-selected players repeatedly select characters to play from the pool of 8 characters described in chapter 5, and a player is chosen to play first at random, referred to as the "active" player. That player selects a random valid move to make. The active player alternates, and the process repeats, until such time as an active player starts their turn with both of their characters fully depleted of health. The player with remaining characters is the victor. When used to generate datasets for analysis, another game is started by picking another pair of random players and starting a game between them; this continues until a predetermined number of games has been played. After a sufficient number of games are played, analysis of the generated datasets is performed. [3] Decisions made by players in the model are random; this is because the model is designed to avoid informed decisions where possible.

[2]Incidentally, this structure allowed a flexible and natural implementation of a procedural simulation containing concepts common in software engineering (such as contexts) and environments (found in simulation frameworks). We imagine that it is easily adopted in existing simulation frameworks such as SimPy[Mat08]. Some additional detail is included in appendix A.

[3]Dataset analysis is explained in section 6.5.

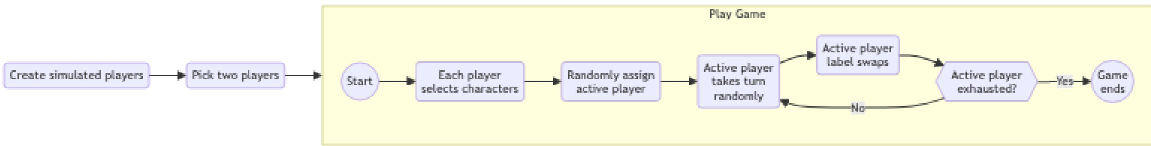

Figure 6.1: A diagram of the “naive model” of RPGLite play used in experiments.

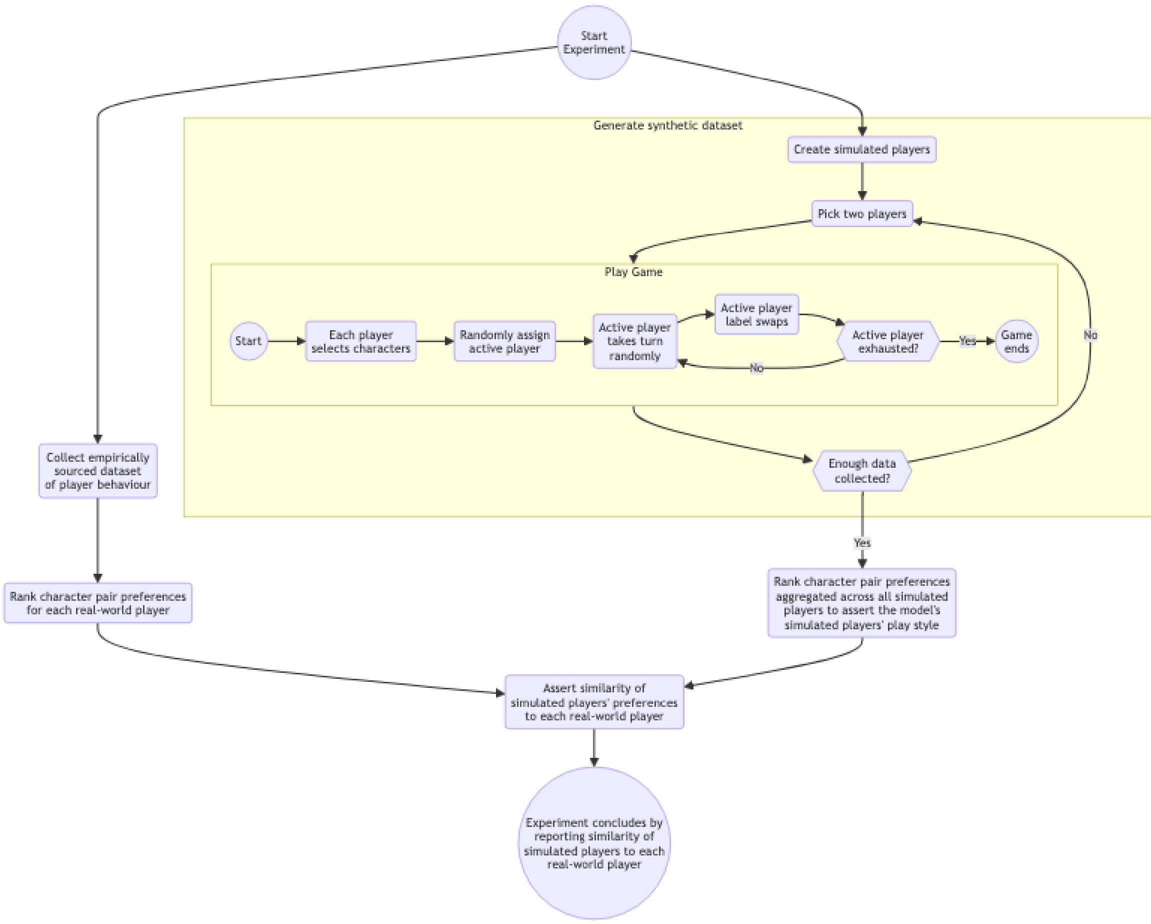

Figure 6.2: A diagram illustrating how the naive model is used to generate datasets used in experiments.

This model illustrates the steps taken by players and disregards whether the simulated players' choices reflect realistic ones. Informed decisions are woven as aspect-oriented changes to behaviour.

There's more I can and should write here. For example, how are seasons represented? How do we switch between them? I also have classes representing individual characters etc…

6.3 Modelling Learning

The naive model makes choices randomly. Informed decisions are made by weaving advice into the model which changes player behaviour. Experiments are evaluated through the distribution of character pairs found in gameplay data, so aspects are required which change that distribution of character pairs. To change the distribution of character pairs chosen, an aspect-oriented model of learning is created which alters player behaviour to select character pairs at the beginning of a game by inferring strong characters from those observed to win games previously. This section describes its design.

6.3.1 What Players Learn

Players learn many things when playing a game. Other than learning about characters and their strengths and weaknesses, players also learn how to take successful turns by making moves strategically. Character pair selection is the focus of these experiments because they are simpler and have a smaller state space for players to understand: the "metagame" for character pair selections is the simpler of the two.

A metagame is a community's perception of "good" gameplay at a given point in time. For example, if certain characters are popular, players may select other characters which are not likely to win against *most* characters but are likely to win against *currently popular* ones. As a result, players may select weak characters strategically, and strategically selected characters change over time in response to changing popular choices in the community. Similar reasoning applies to move selection. For a more thorough discussion of the concept of a metagame, see the survey of literature and definitions by Kokkinakis et al. [Kok+21] or the original but more theoretical work on the topic by **??**.

A design consideration of RPGLite is that its metagame is “solvable”, meaning that there exists an objectively optimal choice for a player to make in any state [Kav21]. As the space of possible moves is much larger than the space of possible character pair choices players are expected to learn optimal choices more quickly, which would lead to an unchanging (“stable”) metagame. In practice, the character selection metagame converged on optimal character selections, whereas players consistently made errors when selecting optimal moves [KM21]. As players’ choices of character pairs is therefore easier to model (and more consistent between players), a model of learning is applied to their character pair selections instead of their move selections. Player behaviour is altered to select objectively optimal moves in every state instead of this being learned over time; this is described in section 6.4.1.

6.3.2 Literature regarding Learning

Different people learn in different ways. Indeed, no universally-accepted definition of learning appears to exist. This is presumably because it is convenient to define what it means to learn differently in the context of different pieces of work. For example, cognitive models of learning can be useful when considering mental processes specifically, whereas functional models of learning lend a more empirically applicable perspective. What it means to learn is outwith the scope of this research; however, the experiments presented in chapter 7 include a model of learning. To justify our model, we consider a functional approach to learning, as this is more closely linked to the empirically focused work of simulating real-world behaviour than cognitive alternatives.

Lachman [Lac97] summarises standard definitions of learning as “[…] a relatively permanent change in behavior as a result of practice or experience”. They observe that these definitions have practical shortcomings such as a focus on behavioural change (as learning may not change behaviour) or conflating learning’s process and its product (the process by which we learn is not obviously identical to its result, of which behavioural change is an example). They suggest learning might be better defined as:

> […] the process by which a relatively stable modification in stimulus-response relations is developed as a consequence of functional environmental interaction via the senses […] rather than as a consequence of mere biological growth and development [Lac97].

They note that their definition distinguishes learning from phenomena such as injury, changes to one's maturity, or sensory adaptation, incorporates stimulus-response relationships the research community consider as learned, and differentiates learning's process and product. Their model is inherently functional, making it useful for the purposes of simulation and modelling, although they offer only a definition of learning and a brief comparison to the standard textbook definition they introduce. The work presented is not intended to demonstrate its improved model of learning empirically, only to discuss its semantic merit. However, the models proposed in this thesis require only a theoretically informed, sound basis for their model of learning, and a lack of empirical justification is not a barrier to the relevance of the model Lachman proposes.

De Houwer, Barnes-Holmes, and Moors [DBM13] propose a functional definition of learning which is primarily concerned with providing a definition of learning which is both accurate and useful for the purposes of cognitive learning research. Doing so attempts to provide a model around which some consensus can be reached; learning is a central concept in psychology, and they describe their definition as supportive of cognitive work without requiring a cognitive model. They introduce their definition as:

> Our definition consists of three components: (1) changes in the behavior of the organism, (2) a regularity in the environment of the organism, and (3) a causal relation between the regularity in the environment and the changes in behavior of the organism.

This model of learning contains more nuance than the "textbook definitions" of learning they paraphrase as "a change in behaviour that is due to experience" but does not stray far from the core concept of an environmental stimulus impacting behaviour in a causal fashion. The introduction of "regularity" to their definition refers to the presence of the stimulus with some form of repetition, either through multiple instances of a stimulus at different times or the same stimulus occurring concurrently. De Houwer, Barnes-Holmes, and Moors [DBM13] observe that their model is straightforward without the sweeping inclusivity of the simple model mentioned earlier and is easily verified (although, as in the work of Lachman [Lac97], empirical verification is omitted in favour of semantic analysis).

Aside from other benefits more particular to their research community, these benefits are especially useful from the perspective of modelling learning in the case of RPGLite. A simple functional definition can be captured in a software model and introduces fewer opportunities for misunderstanding or misapplication than a more complex or theoretical model. It also introduces concepts such as regularity and causality other definitions do not. We therefore adopt this definition as a basis for our model of learning.

6.3.3 Modelling Learning Character Pair Selection in RPGLite

This describes the design of the learning model and it's *pretty lengthy* — it's edited to at least make sense, but it could be shorter if I had the time to rework it entirely.

We use De Houwer, Barnes-Holmes, and Moors [DBM13]'s definition to define an aspect-oriented model of learning that can be applied to the naive model of RPGLite. Their definition of learning gives criteria that this model must meet: players' behaviour must be influenced by their learning to meet the definition's first criterion; repeated experiences in successive games must influence the direction of players' learning is required by the second; and the third criterion requires that this must happen in a causal manner. To meet these, the learning model's design should model a causal relationship between a player's observation of successful character pairs and their future choices of character pairs.

To fulfil these requirements, a model might draw on previously successful character pairs to determine future ones. One approach is to model learning as consistently playing the character pair which most *recently* was observed to win a game. A completed game must have a winning pair,[4] and we can select this pair when playing future games until a different pair is observed to win instead. However, this does not align with one's intuition of how players *would* engage with a game in the real world. A player seems unlikely to be deterred from a strategy they believe is ideal when RPGLite's random nature gives them an outcome they could percieve as unlucky. We can expect players to understand that *perfect* play might not be *winning* play: in some games, the right moves might not lead to a successful outcome due to moves randomly missing opponent characters. Equally, players may take time to become confident in a strategy. We would expect a player to explore character choices before settling on a preferred pair early in their experience, and would expect experienced players to choose characters based on what they have learned through their experience rather than continuing to explore their options. From these observations, we can see that:

① There are scenarios where players can be expected to observe wins/losses *without* incurring

[4] Or be forfeited, in which case the previously winning pair could be substituted.

behavioural change.

② Players' confidence in what they have learned can affect their inclination to rely on that knowledge when making decisions.

③ What players learn in successive games would have a small impact in their early experiences, but an increasingly significant impact proportional to their experience in the game.

The model of learning used to simulate RPGLite players can be explained following a similar structure: ① implies that players' observations regarding winning characters is separate from behaviour change; ② implies some mechanism determining players' inclination to use their knowledge when choosing characters (rather than exploring their options); and ③ implies that their inclination to rely on their knowledge instead of exploring the state space is proportional to the amount of experience players have.

To fulfil requirement ① and separate win/loss observation from behaviour change, a player's assessment of how likely a character pair is to win a game is represented as a probability mass function (*"PMF"*) updated through its own aspect (which is described in more detail in section 6.4). The PMF maps character pairs to their chance of being selected by a player, and initially tracks all character pairs as having an equal chance of being selected. After every game, the chance of selecting the winning character pair increases, and the chance of selecting any other pair decreases. The sum of probabilities of being selected across all character pairs always sums to 1 (100%). This is implemented as a record of the winning character pairs observed by a player: many ways of producing a PMF from a sequence of wins exist, but for the purposes of explanation, one such method is to take the proportion of wins for every character pair as their probability of being selected. This method produces a valid PMF because the sum of those proportions accounts for 100% of observed wins and losses, so the sum of all probabilities must also be 100%.

Requirements ② and ③ are fulfilled through a separate mechanism to control whether players use this PMF to make decisions. Players are expected to explore possible options early in their experience, and rely on their observations when they have completed many games. This is because Lachman [Lac97] identifies that the experience of a learning agent draws from "regularity" in their environment, and requiring many games to be played before players' behaviour changes ensures that a regular experience is present.[5] The mechanism controlling whether players will make decisions based on previous experiences is referred to as "confidence", referring to their confidence in their experiences at a point in time. A model of confidence fulfils requirement ②, and ③ is fulfilled by confidence increasing proportionally to players' experience.

Confidence is modelled in these experiments as a monotonically increasing function mapping experience (quantified as games played) to confidence as a percentage chance 0% and 100% that a player determines their character choice based on their observations of wins and losses rather than exploring the space of possible choices. Players are "confident" when making a decision with the probability determined by their confidence model.

If the player is not confident, their behaviour is unchanged and they select character pairs randomly as in the naive model. If they are confident, their behaviour is instead informed by their experiences and they select a character pair from the distribution defined by the PMF that their historical observations define. As the PMF affords higher probabilities to repeatedly winning character pairs, player behaviour is causally affected by the regularity of their experience. This implements a realistic model of learning in fulfiling the requirements of the functional model proposed by Lachman [Lac97].

6.3.4 Modelling Player Confidence

To model confidence we require a function mapping the number of games a player has completed to a probability between 0 and 1, fitting the criteria described in section 6.3.3. A sigmoid curve fulfils these requirements: it rises monotonically and produces values between 0 and 1. It also notionally conforms to an intuition around "confidence" as a behavioural trait: like a sigmoid, players' confidence starts low and remains so until it reaches some inflection point, after which one's confidence increases more significantly, with the rate of this increase tapering as experience continues to increase. However, not all real-world players might express the same traits in their growth of confidence, and this intuition is not universally applicable: the shapes of players' confidence sigmoids differ in the real world. To answer the proposed research questions, an ideal model of confidence is not required, but one which

[5] Note that the regular experience might be one of the character pair choice not having an impact at all; if the player observes all character pairs winning an equal number of times, the PMF would reflect this and player behaviour would effectively remain random. However, the player would choose each character pair at an equal rate because they would have learned that the choice was inconsequential, so even in this scenario learning occurs.

reflects *some* real-world players *is*. To account for this, the confidence model uses a sigmoid which can be parameterised to alter its shape.

A sigmoid curve is suitable for modelling confidence where other curves are not, because we require a period where players lack confidence and explore their options before growing in confidence and remaining at high confidence thereafter, implying the shape of a sigmoid. They are also commonly used in simulation & modelling. Sigmoid curves such as the logistic [Ver45] or Gompertz [Gom15] are widely used when modelling systems [WJ97], but while they fulfil the role of a monotonically increasing curve with asymptotically low and high initial and final states, the shape of such curves is not trivially modified to fit different players' learning styles. To fulfil the confidence model's requirements it is necessary to find an alternative: different players are expected to exhibit more bullish or timid styles of play, so the curve should be parameterised to account for the behaviours of those individuals.

More flexible asymptotic curves were developed by Richards [Ric59] drawing on growth curves developed by Von Bertalanffy [Von38], which afford a natural pattern of growth. Richards amends this curve to offer a parameterised growth rate. This can be made equivalent to other curves, including the logistic and Gompertz [FT+84]. This curve allows for a parameterised rate of growth but lacks parameters controlling the points at which growth occurs most rapidly. The relative rate of confidence gain is a separate concern to the point at which such growth occurs: a player might cautiously grow in their confidence until they are already very experienced, or might bullishly grow in confidence yet plateau early, taking longer to reach complete confidence in themselves than they did to garner an initial increase, regardless of their relative growth in confidence.

The flexibility of a parameterised relative growth rate appeals to the notion that different players would gain confidence at different rates, but the point at which confidence accelerate most must also be controlled. We therefore employ the Birch curve, proposed by Birch [Bir99] for its increased flexibility as compared to the Richards curve combined with its additional parameter used to control the curve's shape. [6] Different players might exhibit different rates of growth in their confidence, and might grow maximally in their confidence at different points in their experience. It is defined by the equation:

[6] Birch [Bir99] refers to shape to mean the point of inflection of a curve. The point of inflection of an exponential rise to a limit is at its initial point; the point of inflection of the logistic curve is in the exact midpoint of the curve's growth.

$$\frac{dy}{dt} = \frac{ay(K-y)}{K-y+cy}$$

Where c is its curve parameter, K is its upper asymptote[7], a is its relative growth rate (RGR)[8], and y is the value of the curve at a given point in time. The birch curve can represent other curves through its shape parameter: at $c = 0$ the curve is exponential, and at $c = 1$ the curve is logistic [Bir99]. As the birch curve models the properties of different players' confidence through its shape parameter it is a suitable candidate for the model of confidence required by the model of learning described in section 6.3.3. Its implementation in the aspects altering the naive model is given in section 6.4.2.

6.4 Aspects Applied

To generate datasets for experiments the naive model is augmented by weaving aspects. These aspects fulfil different functions and can be categorised as either: aspects which instrument the model to simplify experimental observation and prepare the model for use in experiments; aspects which instrument the model to observe its state, assisting the implementation of aspects which change behaviour; and aspects which alter players' behaviour.

Aspects which alter the model to make it appropriate for use in the experiments are described in section 6.4.1. These include altering the moves made to more closely resemble real-world move selection and handling edge cases which that change introduces. Aspects which instrument the model to observe its state are described in section 6.4.2. Aspects implementing behaviour change including the learning model are described in section 6.4.3. A diagram of a game of RPGLite's naive model annotated with aspects and their join points is presented in fig. 6.3.

[7]This is fixed at 1 for the confidence model, as confidence should never exceed 100%.

[8]The RGR is a common parameter of sigmoid curves and defines the rate at which the sigmoid increases near its inflection point

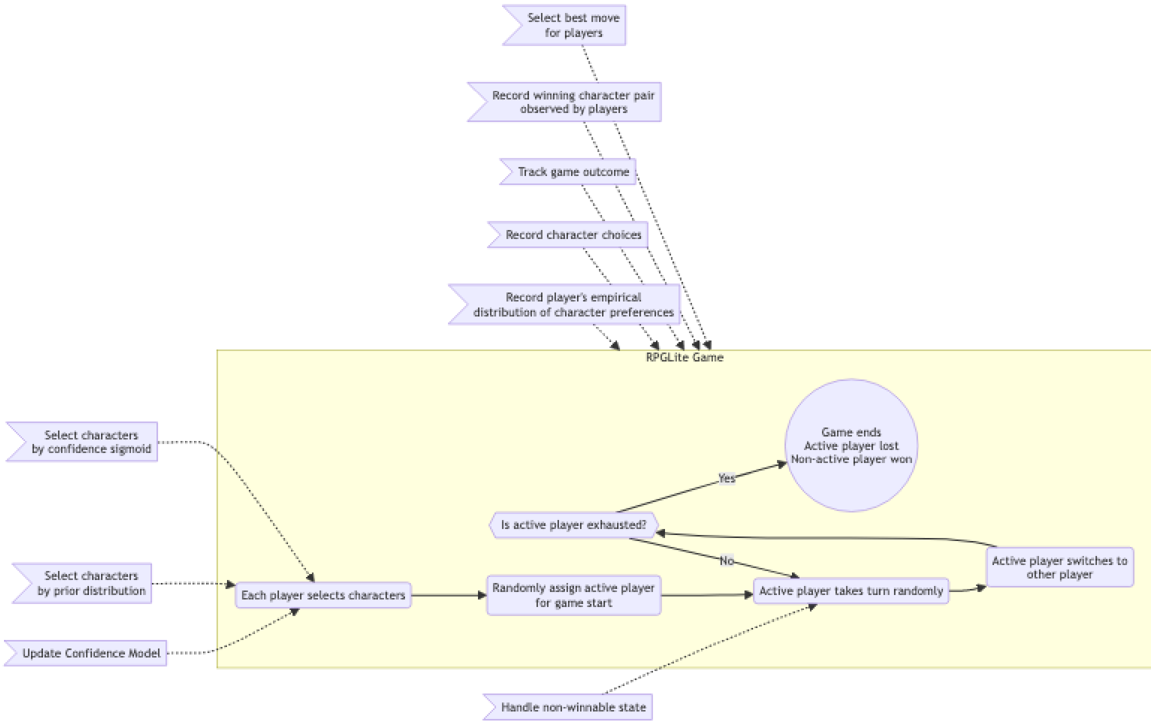

Figure 6.3: A flowchart describing a simulated game of RPGLite, and all aspects woven into the game to implement the various models of learning. Some aspects should not be woven together in the same experimental run, as they implement different models of learning.

6.4.1 Aspects for model improvement

Ensuring the Best Move is Played

Experiments model players' character selection rather than move selection as explained in section 6.3; however, in the naive model players randomly select moves. This is liable to place players in unrealistically weak positions, as players are unlikely to make obviously poor moves such as skipping a potentially useful turn. This is a concern when modelling players learning to select characters because the learning model relies on a causal relationship between what is observed (characters which most reliably win games) and behavioural change (players choosing those characters). If selecting random moves causes simulated players to lose games when they would have won them when selecting moves realistically then move selection would affect character selection. Realistic character selection and move selection are therefore related.

Players of RPGLite usually selected moves optimally. In the majority of cases Kavanagh and Miller [KM21] found that players chose the best move available to them: *"[...] the majority of actions taken were optimal, with a cost of 0.0. In total 73% of the player's season 1 actions and 77.8% of their season 2 actions* *were optimal."*[9] Move selection can be realistically modelled in around $\frac{3}{4}$ of cases by selecting the best move at every opportunity. This behaviour is implemented in an aspect by performing a lookup on the dataset of action costs defined by Kavanagh and Miller [KM21] and selecting the known-optimal move in every case. This is woven into the naive model's `get_moves_from_table` function, which looks up moves from the table of valid moves produced by Kavanagh [Kav21]. It is invoked after the function runs and alters its return value to include only the optimal move, rather than all possible moves. Its source is included in fig. 6.4 as an example of the aspects implemented for this thesis. Include a reference to the repo for the experiments' source code so folk can see the other aspects too.

```python
def choose_best_moves(target, ret, *args, **kwargs):
    '''
    Replaces a set of possible moves from base_model.get_moves_from_table with
    the single best move, forcing that to be taken. Args:
        target: base_model.get_moves_from_table ret: the list of best moves to
        be taken at the game's current state *args: args for the function
        **kwargs: keyword args for the function

    Returns: a list containing only the best move of all moves in ret
    '''
    gamedoc = args[0]

    if list(map(str, gamedoc.get('moves', [None, None])[-2:])) == ['skip', 'skip']:
        return ret

    sorted_moves = sorted(ret.items(), key=lambda move: -move[1])
    return {sorted_moves[0][0]: sorted_moves[0][1]}
```

Figure 6.4: An example of the implementation of an aspect for developing experiments around the naive model. This aspect ensures that the optimal move is selected by replacing the return value of `get_moves_from_table` with the optimal move the function returned.

The aspect selecting optimal moves handles another concern of move selection. As simulated players' behaviours are augmented to select universally optimal moves, an anomaly in the dataset was identified. In some games, optimal play results in infinite loops: players can arrive in a state where they would skip turns mutually and indefinitely.

This occurs because barbarians deal more damage once they lose a sufficient number of hit points. As a result, dealing damage to a barbarian can result in their having an opportunity to win in one move rather than two in certain health states. Both players can concurrently exist in this state. Therefore, both players' optimal move is to skip their turn. Kavanagh and Miller note that such states were

[9] Different seasons of RPGLite are discussed in more depth in section 5.3.

reached in 64 cases in real-world play but real-world players never skipped their turn. To better mimic real-world play, the aspect returns random moves if the previous two moves in a game skipped. Such a state indicates that the game reached a state where neither player would deal damage to the other when making optimal moves.

Handle Game States with no Viable Moves

A consequence of playing games using only optimal moves is that unexpected states can occur: the dataset of possible moves produced by Kavanagh [Kav21] maps game states to moves a player may make, but not all states exist in the dataset. This is because the dataset was produced through model checking with the aim of identifying the cost of an action regards its impact on a player's chance of winning. If a loss is guaranteed, all moves have a 0% chance of winning; the model checker which produced the dataset of available moves therefore identified that the outcome of the game is already determined in those cases, and produced no possible moves for them.

The winning player in this situation is the player who made the previous move, as the now-active player has no moves with a chance of winning above 0%. This logic is added to the model through an aspect which handles exceptions raised by move selection, catching errors caused by an invalid table lookup. After identifying that the exception indicates that the state does not exist (in this case the relevant exception is Python's `KeyError`), the aspect handles the exception by assigning the losing player's characters 0 health and swapping active players. The game proceeds with the losing player taking the following turn. As this turn starts with the active (losing) player having 0 health, the game ends as expected.

6.4.2 Aspects for Instrumentation

Update Model of Confidence

As described in section 6.3.4, confidence is modelled as a Birch curve. This aspect updates a player's level of confidence using the birch curve when a game ends. It is woven as a prelude to the naive model's `choose_character_pair` function, so that confidence is updated for each player before it is used for character pair selection.

The aspect updates a value in the model's environment which represents the confidence of the player selecting a character pair. The aspect uses the model's environment to find parameters such as players' initial confidence or the value of the curve's shape parameter c. The implementation of the aspect updating the confidence model is given in fig. 6.5. This implementation is edited for readability, as the version used when running experiments contained debugging code and additional logic used by other experiments which were attempted but abandoned.

```
def update_confidence_model(_target, player, _game, environment):

    sigmoid_initial_confidence = environment['special vals']['sigmoid initial confidence']

    if 'confidence' not in environment:
        environment['confidence'] = dict()
    y = environment['confidence'].get(player, sigmoid_initial_confidence)

    k = 1  # upper asymptote
    a = environment['special vals']['rgr']

    if environment['special vals']['sigmoid type'] == "birch logistic":
        c = 1  # for logistic curve
    elif environment['special vals']['sigmoid type'] == "birch exponential":
        c = 0  # for exponential curve
    elif environment['special vals']['sigmoid type'] == "birch controlled":
        # the curve value set specifically
        c = environment['special vals']['birch c']
    else:
        # birch, but not specified which one
        raise Exception("Birch equation used, but no specific equation requested")

    # Update value of player's confidence using Birch (1999)
    environment['confidence'][player] = y + a * y * (k - y) / (k - y + c * y)

    return environment['confidence'][player]
```

Figure 6.5: An example of the implementation of an aspect tracking model state. This aspect updates the value of a player's confidence model and is woven as prelude to the naive model's `choose_character_pair` function.

Record Prior Distribution of Character Preferences

To implement the prior distribution model, the character pair preferences of the player being simulated must be calculated. The prelude aspect implementing this is shown in fig. 6.6. Computing this distribution involves loading, parsing, and filtering the entire dataset of RPGLite gameplay data. However, the program running experiments must load this dataset to construct the parameters controlling the simulation — in particular `training_data` and `testing_data`, introduced later in section 6.5.1 — so to avoid duplicating this work, the values are retrieved from the stack frames of the function which constructed the model parameters. The players being simulated and games played by them are identified by their variable names in the stack frame, and their distribution is calculated through a convenience function, `find_distribution_of_charpairs_from_players_collective_games`, the implementation of which is given in fig. 6.7.

```python
def prelude_before_learning_by_prior_distribution(target, actor, gamedoc, env, *args, **kwargs):
  # Before we fuzz, we need to make sure the prior distribution is set up in the game's environment.
  # Calculate and inject it if it's not already there.
  if 'simulation prior distribution' not in env:
      # Grab players and games from an earlier stack frame
      # (avoids weaving an aspect to collect the info)
      frame_to_find_name = "generate_charpair_distributions"
      for frameinfo in inspect.stack():
          if frameinfo.function == frame_to_find_name:
              break
      games = frameinfo.frame.f_locals['games']
      players = frameinfo.frame.f_locals['players']

      from experiment_base import find_distribution_of_charpairs_from_players_collective_games
      env['simulation prior distribution'] = find_distribution_of_charpairs_from_players_collective_games(players, games)
```

Figure 6.6: The prelude aspect calculating the distribution of character pairs selected by the real-world player being simulated, which is reproduced by simulated players using the prior distribution model.

This allows the calculation of the distribution to be separated from the change to players' character pair selection. Separating these into two different aspects allows them to use different join points: this aspect is a prelude and operates before character pairs are chosen, whereas the change made to character selection is performed using a "within"-style fuzzer from PyDySoFu. The `if` statement at the beginning of the aspect, as shown in fig. 6.6, avoids calculating the distribution more than once.

Bit thin maybe? I'm not sure what to say about the

```python
def find_distribution_of_charpairs_from_players_collective_games(players, gameset):
    distribution = dict()
    for player in players:
        player_distribution = find_distribution_of_charpairs_for_user_from_gameset(player, gameset)
        for charpair, count in player_distribution.items():
            distribution[charpair] = distribution.get(charpair, 0) + count

    return distribution

def find_distribution_of_charpairs_for_user_from_gameset(player, gameset):
    charpair_distribution = dict()
    for game in gameset:
        # Game is from RPGLite's real-world mongodb if it has a `_id` field.
        if '_id' in game:
            game = convert_gamedoc_to_tom_compatible(game)

        if player in game:
            char1 = game[player]['chars'][0][0]
            char2 = game[player]['chars'][1][0]
            pair = char1 + char2 if char_ordering[char1] < char_ordering[char2] else char2 + char1
            charpair_distribution[pair] = charpair_distribution.get(pair, 0) + 1

    return charpair_distribution

def convert_gamedoc_to_tom_compatible(gamedoc):
    '''
    We use this to convert a game document from MongoDB to the kind generated by the simulation, for ease of comparison.
    '''
    new_gamedoc = deepcopy(gamedoc)

    new_gamedoc['players'] = gamedoc['usernames']
    new_gamedoc[gamedoc['usernames'][0]] = dict()
    new_gamedoc[gamedoc['usernames'][1]] = dict()
    new_gamedoc[gamedoc['usernames'][0]]['chars'] = [gamedoc['p1c1'], gamedoc['p1c2']]
    new_gamedoc[gamedoc['usernames'][1]]['chars'] = [gamedoc['p2c1'], gamedoc['p2c2']]

    return new_gamedoc
```

Figure 6.7: The implementation of a helper function which calculates the distribution of character pairs from the games a player completed.

Record Character Pair Choices

It is important to keep track of the characters chosen by simulated players, as this data is required to analyse the outcome of the experiments outlined in section 6.1.2. However, the naive model makes no special accommodation for the collection of character choices made by players. Character choices could be calculated by iterating through the list of completed games after a simulation is complete, filtering for games relating to a specific player. However, the collection of character selection data can be simplified by tracking what was played at the end of a game. This is achieved with with an aspect which observes the character pairs played in a completed game, and appends each players' chosen pairs to their own list. This aspect is applied after its join point executes (an "encore" in PyDySoFu parlance reference pdsf's explanatory chapter / re-engineering chapter here? unsure which is most appropriate / introduces the concepts first.).

There are many ways to collect this data, aspectually or otherwise; however, the simplicity of collecting information mid-process without requiring modification of a base model demonstrates the flexibility of an aspectual augmentation of models and simulations as an approach. Collection of chosen character pairs mid-simulation is an opportune example of this convenience.

Track Detailed Outcomes of Games

Similarly to the recording of character pair choices, the outcomes of games must be tracked. This information is important as an input of our model of learning, as successes and failures observed by players for each character pair can inform their models of learning. Some models of learning implemented, such as one including a hyperbolic decay bias (to be explained later, in **??**), require not only knowledge of the proportion of wins each character pair has achieved, but also the history of wins and losses: players might be biased toward recent win/loss observations, discounting older experiences in favour of the new. Detailed historical win/loss information must therefore be recorded.

As with character pair choices, the naive model was not engineered with the intention of providing this information specifically. However, the model is easily instrumented to collect such information at many suitable points. As with recording a player's chosen character pairs, an "encore" aspect was implemented which records wins and losses observed for character pairs on game end. Specifically, the aspect models players observing character pairs which won at the end of a game, lost at the end of a game, and also pairs which were uninvolved in the game. This requires a list of outcomes (wins, losses, and neither) for every player, for every character pair. The lists of observed outcomes for a given character pair record `True` for a winning pair, `False` for a losing pair, and `None` for a character which was not involved in the game's outcome. Each player observes the relevant state for every character at the end of any game they play. Provisions were made in the implementation of this aspect for a player only paying attention to their own outcomes (i.e. only recording whether their own pair won or lost, without consideration for their opponent's outcome), though in practice all simulations were performed with players observing both their own outcomes and those of their opponents.

Record Winning Pair observed by Players on Game End

The simple model of learning makes use of a distribution of winning teams to simulate learning (the implementation of which is explained in section 6.4.3). Rather than coalescing the more complex dataset collected as described in section 6.4.2 to achieve the desired format, another aspect can be applied to collect the data in a simpler format directly.

An aspect was implemented which models players observing winning character pairs on game end and collects the data in a simple list of character pairs which won games, rather than a mapping of character pairs to game end states as described in section 6.4.2. Character pairs which lose games or are not involved are not recorded for the purpose of the alternative model of learning.

This aspect needs a bit more of an explanation. Maybe a code snippet?

6.4.3 Aspects Implementing Models of Learning

Still editing but the other two subsections have no intro; right now this has three paras!

The aspects described above implement setup for the models of learning applied in this thesis, both instrumenting the underlying RPGLite model for observation and tweaking behaviours of the naive model. Two aspects remain unexplained, implementing the learning models themselves.

The first model of learning left to be described is implemented as described in section 6.3.2, drawing from a record of character pairs observed to win in earlier games to select characters in future games, in accordance with a model of confidence which activates either this behaviour or a randomly selected pair. A detailed description follows in section 6.4.3.

The second model of learning left to be described implements a similar logic, but introduces a weighting on the history of winning pairs a player observes, introducing a hyperbolic discounting bias to historical observations. In this model, players increasingly discount old observations in favour of recent ones. A model of learning with a hyperbolic discounting bias is described in section 6.5.

Character Selection using Prior Distribution

The prior distribution model alters character selection by changing the distribution of character pairs selected from when simulating RPGLite play. The distribution they choose from is changed to reflect the distribution selected by the player being simulated. Another aspect, explained earlier in section 6.4.2, calculates that distribution and stores it in the simulation's environment. The role of this aspect is to use that distribution when selecting character pairs at the beginning of a game.

The aspect implementing the prior distribution's behaviour change is shown in fig. 6.8. It is implemented using PyDySoFu's fuzzers (or "within"-style aspects) to insert additional logic within character pair selection rather than prepending or appending logic to it. Logic is added by modifying the target program's abstract syntax tree (*"AST"*), a native representation of Python code. The steps of the fuzzer's target — provided by PyDySoFu as the fuzzer's first argument — are nodes of the AST representing the function definition of the target. The target in this case is `choose_chars`. An implementation of this function is given in fig. 6.9, edited to show how the lines in the function definition map to values in the `steps` argument of the fuzzer.

Python code to select character pairs using the desired distribution is written in a multi-line string and parsed into an AST. This code:

- Takes the distribution of character pairs from the game's environment, represented as a map of two-letter strings (for example, "KA", representing the pair `Knight`, `Archer`) to the number of times a player selected the pair represented by that string,
- Constructs a list, `possible_choices`, which contains the two-character string identifying a

```
def fuzz_learning_by_prior_distribution(steps, target, actor, env, *args, **kwargs):
    get_choices_from_prior_dist_injected_code = '''
players = _context['players']
games = _env['games']

distribution = _env['simulation prior distribution']
possible_choices = list()
for pair, count in distribution.items():
    for _ in range(count):
        possible_choices.append(pair)

char_class_map = {
    "K": Knight,
    "A": Archer,
    "R": Rogue,
    "W": Wizard,
    "H": Healer,
    "B": Barbarian,
    "M": Monk,
    "G": Gunner
}
chars_chosen = [char_class_map[char] for char in choice(possible_choices)]
'''
    to_inject = ast.parse(get_choices_from_prior_dist_injected_code)
    return [steps[0]] + to_inject.body + [steps[2]]
```

Figure 6.8: The implementation of the prior distribution model's change to player behaviour, implemented as a PyDySoFu fuzzer which inserts character pair selection logic into the `choose_chars` function of the naive model to select character pairs using the distribution taken from RPGLite gameplay data in an earlier aspect.

```
def choose_chars(_actor, _context, _env):
    if _actor not in _context: # First "step", or steps[0]
        _context[_actor] = dict()

    chars_chosen = sample(characters, 2) # Second "step", or steps[1]
    _context[_actor]['chars'] = [c() for c in chars_chosen] # Third "step", or steps[2]
```

Figure 6.9: The naive model's `choose_chars` function, which selects character pairs for a player to play a game with randomly.

character pair as many times as the character pair was chosen by a player,

- Creates a map of letters to classes implementing the characters those letters represent (for example, “K” maps to the `Knight` class),
- Sets the `chars_chosen` variable to a list of two classes, by selecting a character pair from the `possible_choices` list at random and adding the class implementing each character in that pair to the `chars_chosen` list.

This leaves `chars_chosen` in the same state as it would have been without the application of the prior distribution model: a list of two character-implementing classes. However, those classes were selected using the distribution produced from RPGLite gameplay data. The AST nodes produced by parsing this code are used to replace the second step of the second step in `choose_chars`. The rest of the function logic remains the same. The changed AST nodes are returned by the fuzzer and compiled by PyDySoFu, ensuring that the modified function definition is executed and so changing player behaviour.

Character Selection using confidence sigmoid

The first model of learning applied through aspect orientation draws on character pairs a simulated player observed to have won games (as recorded by the relevant instrumenting aspect, discussed in section 6.4.2). The record of previously winning character pairs defines a probability mass function by selecting a character with equal probability to their rate of appearance in the history of winning characters.

Selecting a character based on this distribution is gated by a confidence model as described in section 6.3.4. If a simulated player’s confidence model indicates insufficient experience to found decisions on, players will instead select characters randomly, effectively exploring their space of possible choices. By doing so, players have the time and opportunity to observe many matchups between different character pairs. Time to observe matchups is important because some character pairs may only present a strong choice if played against specific alternatives. We can imagine a character pair which is extremely effective against 50% of pairs, but extremely likely to lose games played against the remaining 50% of possible opponents. Without exploring possible matchups, a player may lack observations which would inform them about whether a character pair is effective in general, or in a narrow set of circumstances. A confidence model encouraging early exploration promotes the experience of a wide variety of matchups, avoiding this issue.

Another concern early exploration mitigates is that simulated players need opportunities to build well-rounded priors: if a character pair is never selected by any player, experienced players basing their character pair choices on those which previously win games cannot include pairs which were never selected, because they will never have had the opportunity to win games at all.

This aspect was implemented as an "around" aspect in PyDySoFu's parlance, allowing it to apply additional logic before and after its join point. It could equally appropriately have been implemented as an "encore", applying logic only after its join point had executed; the aspect discards the random selection made by the base model unless the simulated player lacks confidence. Either implementation can return a different character pair choice to the join point's caller, allowing the chosen character pair to effectively be overridden, and so allowing the aspect-applied RPGLite model to proceed executing as it would otherwise; the only difference being the pair of characters "chosen" by the simulated player whose behaviour was augmented.

6.4.4 Answering the First Research Question

First RQ needs addressing here, as we've built the aspects which represent behavioural variations and so can show what those changes to a model look like as advice WRITE ME WRITE ME WRITE ME WRITE ME WRITE ME WRITE ME WRITE ME WRITE ME WRITE ME WRITE ME

6.5 Model Implementation Details

The aspects described in section 6.4 are used to investigate whether aspect-oriented programming can be used to augment models, add new behaviours and parameters to them, and successfully do so across different models. The results of these investigations are presented in chapter 7; some details about the implementation of those experiments are given first in this section.

The parameters used to control experiments are introduced in section 6.5.1. The representation of experiments' results is given in section 6.5.2. The statistical methods used to evaluate those results are explained in section 6.5.3. Finally, the methods used to control simulated players' interactions with their environments to produce reliable datasets are explained in section 6.5.4.

6.5.1 Model parameters

Many parameters are used to control simulated play of RPGLite. They affect the impact of the learning model and the scaffolding which runs simulations, for example by determining the number of simulated players. Parameters which control an experimental run are collected in the `ModelParameters` class, shown in fig. 6.10. Defining these values in a class helps to identify them in the codebase, and defines all variables which are used when optimising the model.

As there are a significant number of parameters and many have a straightforward impact on simulated play, a complete description is given in appendix B in the interest of readability. A description is given here for parameters which have a non-obvious effect, or are the focus of the experimental methodology in section 7.1, or are central to the results given in section 7.3 and section 7.4. Variables used in these ways are:

starting confidence The initial value of the model's confidence curve.

assumed confidence plateau A corresponding high value for the model's confidence curve. As the curve's growth rate is calculated to align with the number of games completed by the real-world player being simulated, and the curve asymptotically approaches 1, a high value is required which represents the expected confidence of a real-world player having played a significant number of games.

curve inflection relative to numgames The proportion of games played by the real-world player being simulated at which the player's confidence curve should reach the parameter `assumed_confidence_plateau`. This allows for control over the growth rate of the curve while linking the rate of growth to the number of games played by the real-world player being simulated;

```python
from dataclasses import dataclass

@dataclass
class ModelParameters:
    c: float
    curve_inflection_relative_to_numgames: float
    prob_bored: float
    boredom_enabled: bool
    training_data: list
    testing_data: list
    assumed_confidence_plateau: float
    starting_confidence: float
    iteration_base: int
    number_simulated_players: int
    advice: list[tuple[str, str, str|callable]]
    players:list[str]
    args:list[any]
    kwargs:dict[str:any]
    boredom_period: int = 25
    initial_exploration: int = 28

    @property
    def boredom_period(self) -> int:
        '''
        Number of games to play before checking player boredom.
        Attempts to allow every player combo to play each other once on
           average before checking again.
        '''
        return int(self.number_simulated_players**2)/2

    def active_dataset(self, testing) -> list:
        return self.testing_data if testing else self.training_data

    def iterations(self, testing) -> int:
        '''
        The number of games to play when generating a synthetic dataset
        '''
        if self.boredom_enabled:
            return self.iteration_base
        return int(self.number_simulated_players**2 * len(self.
           active_dataset(testing)) / 2)

    def rgr(self, testing) -> float:
        '''
        RGR for the parameterised c value, number of games to play, and
           start/end confidences.
        '''
        if not hasattr(self, '_rgr_cache'):
            self._rgr_cache = dict()
        if self._rgr_cache.get(testing) is None:
            num_games_to_confidence = len(self.active_dataset(testing)) *
                 self.curve_inflection_relative_to_numgames
            self._rgr_cache[testing] = \
                curveutils.rgr_yielding_num_games_for_c(
                    num_games_to_confidence,
                    self.c,
                    start=self.starting_confidence,
                    limit=self.assumed_confidence_plateau)
        return self._rgr_cache[testing]
```

Figure 6.10: A class defining the parameters of the model of learning RPGLite which vary when optimising for a given player. For layout & space reasons, some methods of minor significance are removed.

C The curve parameter of the confidence model's underlying birch curve;

prob bored The probability that a player with confidence above `assumed_confidence_plateau` becomes "bored" and stops playing RPGLite. These players are removed from the simulated playerbase and replaced with new players. This mechanism is discussed in further detail in section 6.5.4.

advice A list of tuples defining advice to weave when generating data. Pieces of advice are uniquely defined by a tuple containing the type of aspect to weave (such as `"prelude"` or `"error_handler"`), a string-represented regular expression defining the join point of the advice, and a `callable` to use as an aspect when weaving the advice. As `ModelParameters` instances are serialised to disk to persist experiment results and Python functions are not seralisable, this may also be a string value; strings are IDs of aspects, and are replaced with their corresponding aspect on deserialisation.

training data The set of real-world games used as a training fold when optimising parameters for simulating a player. This is used when annealing toward optimal parameters for the learning model, and so is explained in more detail in section 7.1.

testing data The set of real-world games used as a testing fold when optimising parameters for simulating a player. This is used when annealing toward optimal parameters for the learning model, and so is explained in more detail in section 7.1.

As these determine all variables affecting data generation, individual experimental runs[10] can therefore be reproduced using the class. The convenience method `run_experiment()`, shown in fig. 6.11, is provided to simplify reproducing experimental runs with `ModelParameters` objects. These parameters are varied in some experimental runs to optimise the model and represent individual players' learning to find a combination of parameters which best represent a given player. The strategy for this optimisation is described in section 7.1.

[10]Throughout this chapter, "experimental run" is used to refer to the process of producing a simulated dataset for comparison

6.5.2 Representing the result of an experimental run

As referenced in fig. 6.11, a `Result` class is defined to collect the results of an experimental run and simplify analysis of the distribution of character pair selection produced. The implementation of the `Result` class is given in fig. 6.12. This class is used in the analysis of the success of an experimental run when selecting ideal experimental parameters for the simulation of a real-world player.

A `Result` is not the result of an entire experiment, or the ideal parameters to simulate a given player. Instead, it represents the result of attempting to generate realistic data for a player, which is defined by a `ModelParameters` attribute. Several `Result` objects can be compared against correlation statistics and probabilities of circumstantial correlation (the `statistic` and `pval` properties respectively) using the `within_acceptable_bounds` method to select an optimal set of parameters to simulate an RPGLite player. A technique which uses these properties to determine optimal parameters is described in section 7.1.

6.5.3 Quantifying Similarity of Character Pair Selection

To measure the results of experiments in chapter 7, many datasets are generated (with differing model parameters in the case of the learning model) and are compared against a player's completed games to determine which generated dataset is most similar to the empirical one. The distribution of the naive model should exhibit no bias as character pairs are chosen randomly; simulated play with advice woven is expected to exhibit more "realistic" character pair preferences than the naive model as a result of the advice applied. The similarity between a player's character pair preferences and those of simulated players are determined using a rank correlation measurement.

Many rank correlation measurements exist. A non-paremetric measurement is required for measuring the correlation of character pair distributions. Non-parameteric correlation measurements make no assumptions about the distribution of data; the alternative is to use a parametric measurement, which would be selected based on the distribution in the data being compared. The distribution of character pair preferences is unknown, so to avoid assuming a distribution which is universally applicable to all

---

against training or testing folds taken from the dataset of completed RPGLite games. Many experimental runs with different parameters are used to optimise a model.

```python
class ModelParameters:

    # <snipped additional attributes and methods for space>

    def run_experiment(self, testing, correlation_metric):
        real, synthetic = generate_charpair_distributions(\
            rgr_control=self.rgr(testing=False),
            iterations=self.iterations(testing=False),
            games=self.training_data,
            players=self.players,
            birch_c=self.c,
            sigmoid_initial_confidence=self.starting_confidence,
            boredom_confidence=self.assumed_confidence_plateau,
            num_synthetic_players=self.number_simulated_players,
            boredom_period=self.boredom_period,
            prob_bored=self.prob_bored,
            advice=self.advice,
            *self.args,
            **self.kwargs)
        return Result(self, real, synthetic, correlation_metric, testing)
```

Figure 6.11: The `run_experiment()` method on the `ModelParameters` class, which can be used to reproduce an experimental run from its parameters.

```python
from dataclasses import dataclass
@dataclass
class Result:
    params:ModelParameters
    real_distribution:list[float]
    simulated_distribution:list[float]
    correlation_metric:Optional[callable]
    testing: bool

    @property
    def pval(self) -> float:
        return self.correlation_metric(self.real_distribution, self.
            simulated_distribution).pvalue

    @property
    def statistic(self) -> float:
        return self.correlation_metric(self.real_distribution, self.
            simulated_distribution).statistic

    def within_acceptable_bounds(self, pval_threshold:float,
        statistic_threshold:float) -> bool:
        return self.pval < pval_threshold and self.statistic >
            statistic_threshold
```

Figure 6.12: The `Result` class used to collect results of experimental runs and evaluate their success.

players' interaction with RPGLite a non-parametric correlation measurement is chosen.

Two common non-parametric rank-correlation measurements are Spearman's ρ [Spe04] and Kendall's τ [Ken38]. The differences between these methods can affect measurements of character pair preference. Spearman's ρ accounts for the magnitude of difference between ranks, and is calculated as $\rho = 1 - \frac{6\sum(d_i^2)}{n(n^2-1)}$, where d_i is the difference in the rank of the i^{th} observation in each dataset and n is the total number of datapoints in a dataset. Kendall's τ accounts only for their ordering. It can be calculated as $\tau = \frac{C-D}{C+D}$, where C is the count of pairs with the same rank (*"concordant"*), and D the count of pairs with differing ranks (*"discordant"*).

To choose an appropriate measure for comparing character pair distributions, they can be related to potential patterns in those distributions. As a player is likely not to select many character pairs once they have developed preferences (if at all), a slight increase in selecting uncommon character pairs within a simulation could incur a significant increase in the rank of those pairs. This would have a more significant impact on ρ than on τ, as the larger difference in rank is squared in the calculation. Character pairs' rankings may also have small differences between the two distributions but similar positions relative to each other. For example, most character pairs could have ranks only 1 position apart between the distributions. This would have an outsized impact on τ, as relative positioning affects concordance, but the square of a small difference in rank remains relatively small, meaning that ρ would be less affected by this pattern. A choice of correlation metric is thus a choice of what correlation means: it can reflect preference of character pairs relative to each other, or "forgive" small relative preference deviations and impose a greater cost on the scale of disagreement on a given rank instead.

To prioritise the preservation of the order of a ranking character pair preferences are compared using τ. Unfavoured character pairs' ranks might be affected greatly by a small difference in the number of games played, because unfavoured character pairs play few games. A small difference in selection count is therefore more likely affect the rank of those pairs. Minimising rank difference is sensitive to RPGLite's random nature for this reason. Preserving the ordering of character pair selection is also important when considering favoured pairs, as most games are likely to be played with a small number of favoured pairs. Much as large differences in unfavoured pairs' ranks can be caused by random chance, small differences in the ranks of favoured pairs are indicative of a player's preferences. An appropriate correlation measurement must take this into account. Small differences in the ranks of favoured pairs would have a less signifiant impact on ρ than on τ: ρ accounts for the square of the difference in rank, which remains small for small differences. Experimental results are calculated using τ in chapter 7 for this reason: τ prioritises accurately simulating a player's preferences when selecting character pairs, which is the aim of the experiments, and so is the more appropriate measurement.

6.5.4 Controlling State Space Exploration

The model of confidence grows monotonically as described in section 6.3.3, which affects simulated players' exploration of possible character pairs. The learning model uses confidence as a probability that character pairs are selected based on experience. As character pairs which are not played cannot win games, pairs which are not selected for play cannot be chosen by the learning model as it relies on a player's historical observations, which could lead to inaccurate simulation if the unchosen pair is the player's preference. To counter this possibility, two mechanisms are introduced to control state space exploration: "initial exploration" and "boredom".

Initial exploration is a threshold number of games a simulated player must complete before the confidence model determines whether aspects should change player behaviour. The earning model's impact on player behaviour is gated on both a player's level of confidence and the number of games played. This allows players to enough games that later decisions directed by the learning model can make "informed" decisions, having explored the possible options. The length of the initial exploration phase for any player is given as an integer number of games by the `initial_exploration` parameter when generating data.

Boredom is a mechanism which solves a problem introduced by some combinations of model parameters: if the confidence model's growth rate and curve parameter are high, player confidence raises sharply, with the effect that simulated players are unlikely to explore the state space. A situation can arise where the simulation's metagame falls into a local optimum where the entire playerbase collectively observe many historical wins for some character pairs because of a disproportionately high rate of random selection when players explore the space of possible character pairs. This can be overcome by introducing new players into the playerbase over time. New players are guaranteed to explore the state space due to the initial exploration phase. The randomly selected character pairs played in this time mean that veteran players will play games with character pairs which otherwise could not have been selected. This reinforces existing preferences when novel character pairs are weaker than veteran players' preferences, and weakens existing preferences if veteran players are exposed to novel character pairs which are likely to win games played against their preferred pairs. By disrupting the simulation's metagame boredom mitigates potential issues with random play while allowing simulated players to learn over time.

Players are eligible to be removed from the playerbase when their model of confidence reaches a threshold defined by the `assumed_confidence_plateau` parameter. Each player is removed with probability `prob_bored` once this threshold is reached, allowing some confident players to remain in the playerbase for a long time. Any player who is removed is immediately replaced with a new one, so the playerbase has a constant size. The `boredom_period` parameter determines the number of games between these checks, and the mechanism can be disabled entirely using the `boredom_enabled` parameter.

Chapter 7

Results of Experiments concerning Aspect-Oriented Modelling

The naive model of RPGLite and the aspect-oriented models of learning & confidence which are described in chapter 6 are designed to answer the following research questions:

Laurie ▶*i think there is no RQ1 (it's defined in macros.text but not used AFAICS)? fwiw, i tend to give these sorts of macros non-numeric names and then i'd have 'item[RQsomehumantextnumber] RQsomehumantext' so that I can rename/renumber them arbitrarily*◀

RQ2 *Can models of systems more accurately reflect their subjects through advice-based improvements?* Laurie ▶i must admit that i don't have an intuition of what 'advice-based' might mean, and a quick grep through the PDF didn't turn up a definition. in sec 1.2.1, 'advice' is defined as 'aspect + join point', but i'm not sure what's different between 'AOP' and 'advice-based'. they feel like synonyms?◀

RQ3 *Can advice be used to accurately introduce behaviours or parameters into a model which were not originally present in it?*

RQ4 *Can advice be used as a portable module, such that aspect-oriented improvements to one model can be woven into another without loss of performance?*

To investigate each research question, relevant advice is woven into the naive model, and datasets are generated of recorded simulated gameplay. To answer the proposed research questions, these datasets are compared against the empirically sourced datasets described in chapter 5. Different experiments require different comparisons and yield different contributions, but all make use of the same technical foundations described earlier in chapter 6.

This chapter explores the results of the experiments which are enabled by the previous chapter's foundations. Section 7.1 describes how synthetic datasets are generated which to yield statistically significant correlation when simulating a given player, and in particular how parameters are chosen for the learning model to yield those results. Following this are three sections each focused on an experiment to answer a research question. These sections are composed of an explanation of the experiment's design, followed by a presentation of results and an evaluation of those results with regards the research question of interest. Section 7.2 describes an experiment answering the second research question, concerning the use of advice to alter pre-existing modelled behaviour. Section 7.3 discusses an experiment answering the third research question, concerning the use of advice to introduce new parameters and behaviours to a model. Section 7.3 presents an experiment answering the fourth research question, concerning the portability of advice as individual modules to new systems, or to changed instances of the same system. The chapter's contributions are briefly reviewed in its conclusion, section 7.5.

7.1 Methodology concerning the Interpretation of Results

7.1.1 K-Fold Validation

Laurie ▶*i think you'll need to define 'folds' and so on, at least in the sense of giving readers an intuition. i've never heard of this stuff before, and only got a very vague idea of what this section is doing.*◀

RPGLite exhibits randomness, and its random nature affects the datasets generated when running experiments; this may cause correlation to appear by chance, biasing results. As discussed in section 6.5.4, efforts are made to mitigate the impact of random play such as periodically refreshing the player pool and ensuring many games are played. Efforts are also made when interpreting results to control for the influence of random chance.

Simulations are run several times, and the datasets they generate are compared against subsets of

RPGLite player data to measure correlation, minimising the impact of randomness when evaluating results. This is achieved using k-fold validation, and "Leave-One-Out Cross-Validation" (LOOCV) in particular. In k-fold validation, the dataset being compared against is divided into k equally sized partitions ("folds"), which can be combined in different ways to produce datasets for training an algorithm or finding optimal parameters for an algorithm ("training folds") and corresponding datasets for testing the results of each optimisation ("testing folds") [RTL09]. LOOCV creates k pairs of training and testing folds, where testing folds are individual partitions, and their corresponding training folds are the union of all other partitions.

All experiments in this chapter make use of LOOCV to minimise bias in results. RPGLite player data is partitioned, and sets of training and testing folds are produced from these partitions. Experiments generate multiple datasets, comparing the results against different testing folds, and optimising model parameters against different training folds where appropriate. In all experiments $k = 5$ to avoid small training & testing folds and to measure correlation many times, which helps to identify individual biased results.

7.1.2 Identifying Model Parameters yielding Optimally Significant Results

Section 7.2 and section 7.3 describe experiments where parameters for the model of learning are optimised. As this optimisation occurs across many folds, it is possible that many parameters produce statistically significant datasets when compared against a given player. These parameters may also vary between folds, as comparison against different testing folds may show correlation with different datasets. A technique to identify model parameters which produce statistically significant results across many folds is therefore required.

Identifying optimal parameters requires searching across two dimensions of statistical significance reported in experiment results: Kendall's τ correlation metric produces a statistic of correlation (which is also referred to as τ in the context of results) and a p-value describing the probability that the correlation statistic would arise in the case that the null hypothesis for a given experiment is true, as described in section 6.5.3. Statistically significant results must demonstrate a sufficiently high correlation statistic and a sufficiently low p-value; "sufficiently" high or low is defined below.

To identify parameters for a player which reliably produce statistically significant results across folds, a threshold is set for both values of τ and their corresponding p-values. Parameters producing results which meet both thresholds across $> 50\%$ of folds are considered to be significant parameters for simulating a player, as they are more likely than not to produce datasets correlating to real-world play. If no such parameters exist, thresholds are weakened, and the search repeats. Thresholds continue to be weakened until doing so would indicate statistically insignificant results, such as large p-value or a low τ value. If no parameters exist which reliably produce statistically significant results for a player, the player is not accurately represented by the model.

Correlation coefficient thresholds are selected at 0.5, 0.4, 0.3, and 0.2. In behavioural science research, a correlation coefficient of 0.5 is considered strong in practice, 0.3 of medium strength, and anything below 0.2 weak; these values are both suggested as guidelines [Coh13] and found in the distribution of published results in metastudies [Hem03]. As the model of learning used in these experiments predicts human behaviour, typical bounds on high and low correlation from the behavioural science community are adopted for these experiments, and a correlation coefficient below 0.2 is discarded in all cases as insignificant. P-value thresholds are 0.01, 0.02, 0.035, and 0.05. A maximum value of 0.05 is adopted by convention in many research communities [HK11]; other p-value thresholds are selected arbitrarily for incrementally more significant results.

To determine which pairs are searched first, they are ordered by their Manhattan distance from the most restrictive pair, $\tau > 0.5$ and $p < 0.01$. This is calculated by summing the number of steps away from the most restrictive threshold for each measurement: a threshold pair $\tau > 0.4$ and $p < 0.2$, with a Manhattan distance of $1 + 1 = 2$, is considered stronger than a pair $\tau > 0.5$ and $p < 0.05$, which has a Manhattan distance of $0 + 3 = 3$, and therefore is used first when searching for significant results.

Experiments in section 7.2 and section 7.3 use this technique to find “optimal” parameters to model each player with statistical significance, if such parameters exist for a player. The algorithm implementing this search is given in appendix C for reference.

7.2 RQ2: Altering Model Behaviour using Aspect Orientation

7.2.1 Experimental Design

To demonstrate the viability of altering a model by using aspects to describe a change in behaviour, the advice applying the already-known character pair distribution to character pair selection is applied. This forms the first of three experiments in this chapter, addressing the research question:

> *Can models of systems more accurately reflect their subjects through advice-based improvements?*

The research question yields a null hypothesis: *models of systems cannot be altered using advice to reflect their subjects more accurately*. If this were the case, there would be no discernable difference between the real world data's correlation against data from the naive model and data from a model with advice woven. Simulated players in the naive model select character pairs randomly, meaning that — if patterns such as personal preference exist in the real-world data, or the data is not randomly distributed — we would not expect much correlation between the two. If weaving advice produces datasets which do correlate with the real-world data, then the advice would have affected the model to more accurately reflect the player being simulated. We would therefore discount the null hypothesis, and answer the research question affirmatively. If no correlation could be produced as a result of weaving aspects, then the null hypothesis would have been demonstrated instead, answering the research question negatively.

Figure 7.1 illustrates the advice woven into the naive model to alter player behaviour to select character pairs with the same distribution as the real-world data exhibits. As this advice causes simulated players to select character pairs with the same distribution as real-world players, the datasets generated using them ought to correlate strongly with the real-world data: rather than being random, the distribution is expected to be the same. This therefore tests our hypothesis that advice can be used to alter model behaviour to be more accurate. [1]

[1] Later experiments will investigate the addition of specific behaviours, rather than reproducing properties of an already-present dataset.

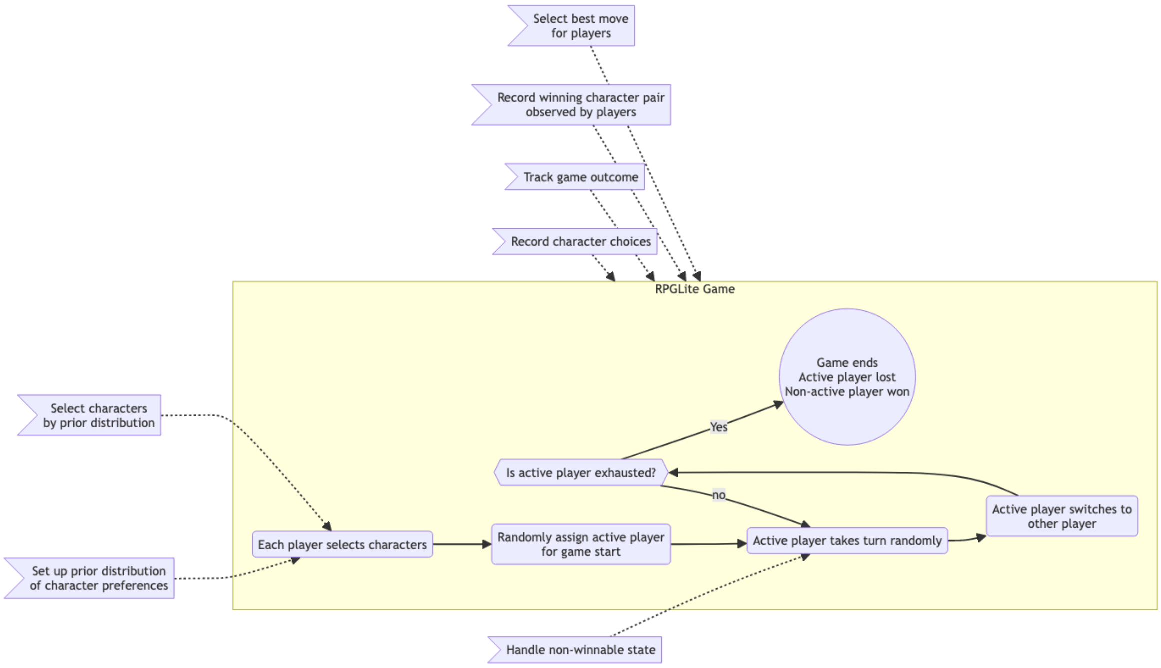

Figure 7.1: Advice woven into the naive model to adopt the distribution used to select character pairs from real-world data.

7.2.2 Results

Results are generated through k-fold validation as described in section 7.1.1. As no additional parameters are added to the model, no searching for optimal parameters is required. A drawback of the lack of searching for optimal parameters is that there is no convenient way to summarise results across folds. However, results are *consistent* across folds; the result of the first fold is given in this section's result tables, described below. Complete tables with data from all folds is given in appendix D in the interest of transparency. Results are shown for players who completed at least 100 games in season 1 of RPGLite, and the experiment is conducted against season 1 gameplay data.

The results of the experiment are shown in fig. 7.2 for datasets produced by the naive model, and fig. 7.3 for datasets produced by applying advice to the naive model which alters the distribution from which character pairs are selected.

Laurie ▶*the double hline above causes formatting issues with superscript numbers. fwiw i am a disciple of the 'booktabs' package, which produces the most beautiful tables in latex i've seen. in particular, it gets horizontal lines 'right'.*◀

Briefly summarised, the only player showing statistically significant results from the data produced by the naive model is from *cwallis*, from whom a p-value of 0.042 and a τ correlation coefficient of

Username	*p-value*	*τ statistic*
apropos0	0.930	-0.013
basta	0.182	0.205
creilly1	0.352	0.141
creilly2	0.764	-0.046
cwallis	0.042	0.309
Deanerbeck	0.881	-0.023
ECDr	0.370	0.135
elennon	0.683	0.062
Ellen	0.417	-0.120
Etess	0.291	0.165
Fbomb	0.169	-0.220
Frp97	0.452	0.119
georgedo	0.185	0.205
Jamie	0.944	-0.0106
kubajj	0.704	-0.058
l17r	0.646	-0.073
Nari	0.330	-0.152
Paddy	0.294	0.166
sstein	0.529	0.101
tanini	0.213	0.185
timri	0.160	-0.220

Figure 7.2: Correlation of real-world character pair selection and those generated by an unmodified naive model.

Username	*p-value*	*τ statistic*
apropos0	6.070×10^{-10}	0.964
basta	6.984×10^{-9}	0.975
creilly1	6.984×10^{-10}	0.961
creilly2	1.154×10^{-8}	0.984
cwallis	2.514×10^{-9}	0.970
Deanerbeck	4.742×10^{-8}	0.979
ECDr	8.455×10^{-10}	0.959
elennon	3.963×10^{-9}	0.973
Ellen	2.538×10^{-9}	0.950
Etess	1.113×10^{-8}	0.994
Fbomb	3.117×10^{-8}	0.996
Frp97	2.440×10^{-8}	1
georgedo	4.719×10^{-8}	0.970
Jamie	5.760×10^{-9}	0.985
kubajj	5.728×10^{-9}	0.966
l17r	1.056×10^{-7}	0.994
Nari	1.965×10^{-8}	0.985
Paddy	1.171×10^{-8}	0.984
sstein	5.017×10^{-8}	0.988
tanini	1.539×10^{-9}	0.952
timri	2.582×10^{-8}	0.990

Figure 7.3: Correlation of real-world datasets of character pair selection and those generated by the naive model with advice woven to bias the characters chosen.

0.309 is produced, indicating medium-strength correlation with a low chance of being a coincidence. As noted in section 7.1.2, a τ statistic above 0.2 paired with a p-value below 0.05 is the threshold set for statistical significance. With a 4.2% chance of being coincidentally generated and 21 players simulated, this outcome is not unexpected. Correlation results from *cwallis'* other folds show no significant correlation, as seen in the complete table presented in appendix D. The results of simulating all players with advice woven show strong correlation and low p-values, meeting the criteria for statistical significance as described in section 7.1.2 in every case.

7.2.3 Answering the Second Research Question

The data shown in fig. 7.2 demonstrates that — without any advice woven into the model to alter its selection of characters — the naive model selects character pairs which do not correlate with those found in real-world datasets at all. As the naive model acts randomly with regards character pair selection, this aligns with expectations.

The data shown in fig. 7.3 contains the correlation of real-world datasets for a player with the naive model, with advice woven to select character pairs from the distribution found in that player's empirical dataset. Every simulated player showed an extremely strong correlation statistic and p-value for their simulated dataset's correlation with their empirical equivalent. While this correlation is extreme, it also matches expectations. Advice which selects character pairs using a known distribution ought to result in that distribution being represented in simulated games.

This demonstrates the hypothesised effect: player behaviour can be altered though the weaving of aspects to produce a model which more closely matches real-world observations in some manner. The research question can therefore be answered affirmatively.

A curiosity of this experiment is that the altered behaviour is not a cross-cutting concern as they are typically defined. Modularisation of these parts of a program is aspect-oriented programming's original use-case [Kic+97], but this experiment shows that the technique can be successfully used in other ways (as suggested by Gulyás and Kozsik [GK99] and Steimann [Ste06]). The success of advice as a mechanism to introduce change to a program, rather than being used as a mechanism to refactor or more elegantly design it, suggests that aspect-oriented programming could be well-suited to exaptation[2] as a tool to introduce new behaviours in a model.

7.3 RQ3: Adding Model Behaviours & Parameters using Aspect Orientation

7.3.1 Experimental Design

The behaviour added to the naive model in section 7.2 impacts how accurately the model reflects a player, but does not model new player *behaviours*. While this is sufficient to demonstrate that aspect-oriented programming can be successfully used to improve a model, it does not answer the third research question:

> *Can advice be used to accurately introduce behaviours or parameters into a model which were not originally present in it?*

I'm not satisfied with this motivation for this RQ and how it's isolated from

[2]Gould and Vrba [GV82] coined the term "exaptation" to refer to a trait of a species which evolved in response to one need, but were later co-opted to satisfy another.

The experiment discussed in section 7.2 changes the distribution used to select characters, but does not parameterise its changes and adds no new activity on the part of simulated players which would require a new parameter. It therefore demonstrates that the software engineering technique can be applied in the context of simulation & modelling, but does not demonstrate that aspect-oriented programming meets the needs of researchers who seek to apply advice which add or make changes to the behaviours of actors in their simulations rather than tweaking existing ones. As a result, answering this research question requires a further experiment, which the aspect-oriented models of confidence and learning introduced in chapter 6 yield.

Aspect-oriented models of confidence and learning are therefore woven into the naive model to produce another in which players learn over time. A player's choice of characters is not selected randomly from a distribution in this model, but informed by their observations of wins and losses in previous games. The advice used is that of the models of confidence and learning described in chapter 6. A diagram of the woven advice is given in fig. 7.4.

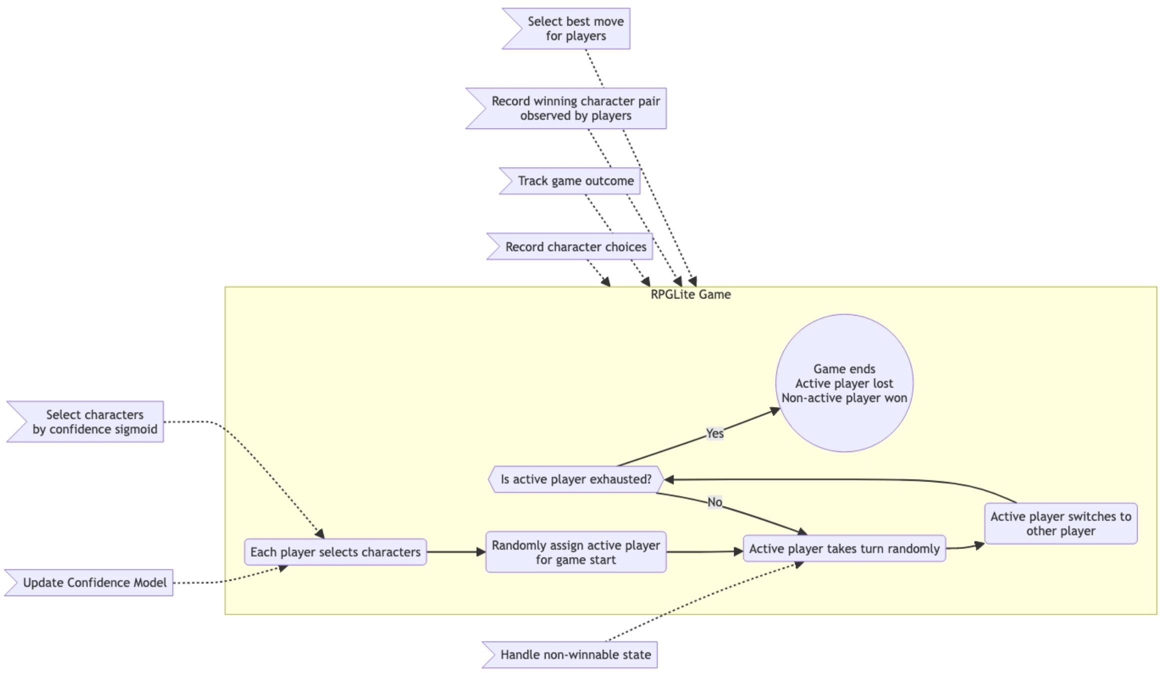

Figure 7.4: Advice woven into the naive model to amend simulated players' behaviour to include learning over time, and to track the relevant parameters for the related model of confidence.

The null hypothesis associated with the research question this experiment addresses is, *advice cannot be used to accurately introduce behaviours or parameters into a model in which they are not already present*. If this were true, the datasets produced by the aspect-oriented model shown in **??** would exhibit no more correlation with real-world data than the naive model does. If a learning behaviour can be successfully introduced, some players' simulations would exhibit consistently realistic behaviour which would correlate with their real-world data.

Unlike the experiment described in section 7.2, this experiment uses no real-world data to inform the actions simulated players make. As a result, if parameters can be found for a player which consistently generates datasets correlating with their empirical dataset, the simulated player's behaviour must have been successfully altered to learn in an accurate manner. This would discount the null hypothesis; the research question could therefore be answered affirmatively.

7.3.2 Results

Results were generated by running the naive model with advice woven which implements a model of learning. Results were generated using k-fold validation and model parameters searched for using the technique described in section 7.1.2. The parameters modified when running simulations were:

- The confidence model's birch curve shape parameter, which could have any of the values $c \in \{\frac{1}{50}, \frac{1}{10}, \frac{1}{5}, 1, 5, 10, 50\}$.
- The coefficient applied to scale the relative growth rate of the confidence model's birch curve, which could have any of the values Laurie ▶to avoid weird formatting (e.g. 'coeff' gets formatted as 'coef f'!) in maths mode, you'll want to put the words in a 'textrm' command◀ $rgr\ coeff. \in \{0.1, 0.3, 1, 3\}$
- The probability that a simulated player would become "bored" and replaced in the player pool for the simulation, as described in section 6.5.4. The probability that a player would become bored could have any of the values $prob.\ bored \in \{disabled, \frac{1}{64}, \frac{1}{16}, \frac{1}{4}, 1\}$

The results have been separated into two tables for legibility: results for players for whom parameters could be found which produced statistically significant correlation are shown in fig. 7.5, and results for players for whom no such parameters could be found are shown in fig. 7.6. For some players, many parameters could be found which produced correlating data; in these cases, fig. 7.5 contains all such parameters. Results are shown for players who completed at least 100 games in season 1 of RPGLite, and the experiment is conducted against season 1 gameplay data.

If there's time, consider making these results tables

Username	*p-value* $<$	$\tau >$	*Confidence curve value*	*Confidence RGR modifier*	*Prob. bored*	*# folds*
apropos0	0.01	0.4	0.2	3	0.062	3
apropos0	0.01	0.4	10	0.1	*disabled*	3
cwallis	0.01	0.4	0.02	0.1	0.25	4
cwallis	0.01	0.4	0.1	0.3	0.25	4
cwallis	0.01	0.4	1	0.1	0.25	4
cwallis	0.01	0.4	10	0.1	0.25	4
elennon	0.035	0.3	50	0.1	0.062	3
Ellen	0.01	0.3	10	0.3	0.016	3
georgedo	0.02	0.3	50	3	0.016	3
georgedo	0.02	0.3	0.1	3	0.016	3
georgedo	0.02	0.3	0.2	3	0.062	3
georgedo	0.02	0.3	0.2	1	0.062	3
georgedo	0.02	0.3	0.02	1	0.062	3
Jamie	0.01	0.4	0.2	1	1	3
Jamie	0.01	0.4	1	3	0.25	3
Jamie	0.01	0.4	5	0.1	0.25	3
Jamie	0.01	0.4	0.02	1	0.016	3

Figure 7.5: Correlation of empirically sourced datasets and those generated by applying the aspect-oriented model of learning.

Simulations of player *cwallis* yielded 26 parameter combinations producing statistically significant results, far more than any other player. For legibility, only parameters which produced statistically significant correlation in 4 folds are shown in fig. 7.5. A complete table with all results is provided in appendix E.

7.3.3 Answering the Third Research Question

?? shows that six player are accurately modelled by the aspect-oriented model of learning shown in fig. 7.4. The answer to the research question is therefore affirmative: advice can be used to accurately introduce new behaviours and parameters to a model which were not previously present.

Not all players are accurately simulated using this model of learning. This result is to be expected. One reason for this is that different players learn in different ways; the model of learning developed in this thesis is not intended to represent some universal model of learning (if such a model can exist) but to represent some players' behaviours, thereby demonstrating that complex behavioural models can be tractably modelled as advice. Another reason is that, as these models draw from historically observed data, learning may be sensitive to a player's personal biases: for example, they may not understand how to usefully employ a certain character, skewing their observations of wins and losses. Finally, real-world players may simply dislike certain characters for superficial reasons, such as artwork or a

Username	*p-value <*	*τ >*	*Confidence curve value*	*Confidence RGR modifier*	*Prob. bored*	*# folds*
basta	N/A	N/A	N/A	N/A	N/A	N/A
creilly1	N/A	N/A	N/A	N/A	N/A	N/A
creilly2	N/A	N/A	N/A	N/A	N/A	N/A
Deanerbeck	N/A	N/A	N/A	N/A	N/A	N/A
ECDr	N/A	N/A	N/A	N/A	N/A	N/A
Etess	N/A	N/A	N/A	N/A	N/A	N/A
Fbomb	N/A	N/A	N/A	N/A	N/A	N/A
Frp97	N/A	N/A	N/A	N/A	N/A	N/A
l17r	N/A	N/A	N/A	N/A	N/A	N/A
kubajj	N/A	N/A	N/A	N/A	N/A	N/A
Nari	N/A	N/A	N/A	N/A	N/A	N/A
Paddy	N/A	N/A	N/A	N/A	N/A	N/A
sstein	N/A	N/A	N/A	N/A	N/A	N/A
tanini	N/A	N/A	N/A	N/A	N/A	N/A
timri	N/A	N/A	N/A	N/A	N/A	N/A

Figure 7.6: Correlation of empirically sourced datasets and those generated by applying the aspect-oriented model of learning for players who showed no correlation, separated for legibility.

lack of interest, which would skew their distribution of character pair selections in a manner which the model of learning would be unable to account for as written.

The accurate simulation of $\approx 28.9\%$ of the sampled playerbase demonstrates two successes of this experiment. First, the fact that model did not accurately simulate *all* players is a positive result: if it did, this would indicate that the experiment was incorrectly conducted in some manner, as biases unrelated to learning are to be expected in players' interactions with RPGLite, making a 100% success rate indicative of an error in the simulation. Second, the research question is answered affirmatively: advice *can* be employed to represent new behaviours within – and add new parameters to – an existing model.

7.4 RQ4: Applying Aspects to New Models

7.4.1 Experimental Design

Laurie ►*i assumed this definition would have appeared at least earlier in the chapter, if not earlier in the thesis?*◄ Aspect-oriented programming is used to produce modules Laurie ►*is 'produce modules' the easiest way to think of AOP? i'd have said it takes in a program and produces another program.*◄. Aspects describe a cross-cutting concern, which can be applied to specific join-points; its use in multiple parts of a program lend it a modular nature.

Advice representing change to a model may only be applied to a single join-point, as is the case for the aspects described in section 6.4, but that advice could be applied to future models. Seen though this lens, the behaviour encoded in a model may not be cross-cutting *within* a model, but cuts *across* models, with each potentially including its own implementation of the behaviour tangled within implementations of other behaviours. Its representation as advice could allow the behaviour to be implemented once, but woven into every model that can make use of it. This is the concept underlying prior research [WS18a] which modelled software engineering teams working under different software methodologies, and used aspect-oriented programming to implement behavioural changes such as distractedness as cross-cutting concerns. However, this research did not confirm its findings against real-world data, and so could not verify that models of behaviours as cross-cutting concerns were accurate.

Chapter 2 proposed the following research question to verify that aspects can cross-cut models rather than modules of a program:

> *Can advice be used as a portable module, such that aspect-oriented improvements to one model can be woven into another without loss of performance?*

As RPGLite was played in multiple seasons, variants of the underlying system were used to collect player data. Changes in game seasons constitute changes to the configuration of the game by strengthening some characters and weakening others. As this alters RPGLite's metagame [Kav21], players are expected to react by re-evaluating their preferred character pairs. This has the effect of creating a second model which differs minimally from the first, which can be used to investigate the accuracy of an aspect cross-cutting two models.

A final experiment is described in this section, in which the model drawing from a known distribution of character pair choices and the model of learning are each applied to RPGLite's second season. The accuracy of these models was examined in section 7.2 and section 7.3 respectively. Should they remain accurate when applied to the second season, we can affirmatively answer the final research question. The research question also yields a null hypothesis: *aspects which are accurate when applied to one model cannot be ported to another*, which can be discounted if and only if the experiment demonstrates aspects being accurately woven into a model of the second season as well as the first.

When an aspect is reused, it is unclear whether values for its parameters which produced accurate data in one model should also be reused. As the learning model introduces more parameters to the model, two experiments can be run using it to investigate parameter reuse. First, the learning model is run using parameters which produced statistically significant data for each player in season 1. Second, the learning model is run using the same methodology as used in section 7.3 to anneal toward optimal parameters using season 2 player data. The methodology for the latter is the same as used for season 1 in section 7.3. In the case of the former, parameters which produced statistically significant data in season 1 are re-run and the datasets produced are compared against relevant player's gameplay data from season 2.

7.4.2 Results

The results of running the model with a change to the distribution used for character pair selection are shown in fig. 7.9. Results are presented in the same manner as with the analogous experiment in section 7.2: results from the first fold are presented in the table for each player, with a complete table available in appendix F. Appendix tables for prior distribution results currently run off the end of their pages. The formatting for these tables needs fixing.

The results of applying the learning model to season 2, parameterised using a player's best-performing model parameters for playing season 1 (found during the second experiment) are shown in fig. 7.7. The results of applying the learning model to season 2, and annealing to discover optimal parameters, are found in fig. 7.8.

Results are shown for players who played at least 100 games in season 2, except for the model of learning run with parameters discovered in the season 1 experiment. As the parameters were found for players who were active in season 1, those players were re-used in season 2. Some players were active in season 1 but played no games; their results are marked as *No Season 2 Data* in the relevant results table, fig. 7.7. As in fig. 7.6, results for players for whom no parameters could be found which produce accurate data in season 1 are marked *N/A* when annealing toward new parameters using season 2 data, as shown in fig. 7.8.

Strange layout issues here where tables don't want to sit on the same pages. Frustrating and leaves the final table a page or two away from this text! Not sure how to fix. Ideally the prior distribution table would come first — currently it's at the end to make the layout slightly nicer. Revisit if you get time. The design of the tables between experiments is diverging. I'm not sure what design will work for them all. Revisit the design; it's important that the results don't look unhinged. Tim may have useful advice here. Laurie ▶*you might want to use the '!' positioning thingy in figure (e.g. 'h!') which nominally 'forces' latex to put tables where you want. in practise, latex still does what it wants, but somewhat less so than without. other tricks you can play include table rotation to get things to fit well (i've done this in several papers) and/or putting one or more tables on their own page, which tends to keep latex happier. in other words, 'h' is latex's least favourite position hint. 't' or 'b' are what i use. 'p' is what latex really wants, even though generally we tend not to what that.*◀

Username	*Season 1 p-value* $<$	*Season 1* $\tau >$	*Confidence curve value*	*Confidence RGR modifier*	*Prob. bored*	*Season 2 p-value*	*Season 2* τ *value*
apropos0	0.01	0.4	0.2	3	0.062	0.597	0.075
apropos0	0.01	0.4	10	0.1	*disabled*	0.440	0.109
elennon	0.035	0.3	50	0.1	0.062	1.0	0.0
Ellen	0.01	0.3	10	0.3	0.016	0.008	0.369
georgedo	0.02	0.3	50	3	0.016	0.419	0.118
georgedo	0.02	0.3	0.1	3	0.016	0.470	0.106
georgedo	0.02	0.3	0.2	3	0.062	0.085	0.253
georgedo	0.02	0.3	0.2	1	0.062	0.097	-0.242
georgedo	0.02	0.3	0.02	1	0.062	1.0	0.0
cwallis				*No Season 2 Data*			
Jamie				*No Season 2 Data*			

Figure 7.7: Correlation for simulation using an aspect-oriented model of learning applied to RPGLite season 2, with parameters identified as optimal for season 1 in section 7.2.

7.4.3 Answering the Fourth Research Question

The fourth research question can be answered affirmatively from the experimental results: fig. 7.9 shows a high degree of correlation when used to simulate season 2 gameplay, and its success is comparable to that of simulating season 1 gameplay as shown in fig. 7.3. Aspects used in the model applying a known distribution of character pairs to players' choices therefore port to other models of RPGLite. As these aspects are written as fuzzers which weave changes within their targets, this result extends to the novel type of advice introduced in PyDySoFu as well as pre-existing types.

While the research question can be affirmatively answered, the results generated using the learning model in fig. 7.7 and fig. 7.8 show little correlation and so add nuance to these findings. Only one player — *ellen* — was successfully simulated using the same parameters in seasons 1 and 2. The same

Username	*p-value* <	τ >	*Confidence curve value*	*Confidence RGR modifier*	*Prob. bored*
Ellen	0.01	0.4	0.02	0.1	*disabled*
aaaa			*N/A*		
apropos0			*N/A*		
basta			*N/A*		
Beccccca			*N/A*		
creilly1			*N/A*		
DavetheRave			*N/A*		
Deanerbeck			*N/A*		
DX13			*N/A*		
ECDr			*N/A*		
Ezzey			*N/A*		
Frp97			*N/A*		
Jhannah			*N/A*		
Luca1802			*N/A*		
l17r			*N/A*		
Martin			*N/A*		
Nari			*N/A*		
sstein			*N/A*		
timri			*N/A*		

Figure 7.8: Correlation for simulation using aspect-oriented model of learning applied to RPGLite Season 2, annealing for parameters specific to season 2.

player was also the only one for whom parameters could be discovered using season 2 data which yielded statistically significant results across a majority of folds. As many fewer players are successfully modelled using these aspects in season 2 than in season 1, these results do not demonstrate that aspects which successfully introduce new behaviours or parameters to one model can be reused across different models. Several factors may contribute to these results.

These results may be influenced by the design of the learning model: it may more accurately represent the behaviour of a player who is new to a game than one who is adjusting to a change in that game. It may be that players respond to a change in RPGLite's metagame by using old strategies and adjusting them as needed, rather than discovering new strategies. The model of learning makes an assumption that simulated players have an initial lack of confidence which impacts how character pairs are selected, forcing players to explore possible character pairs and build preferences over time. If real-world players do not lose much confidence in their ability to select winning character pairs when seasons change then the assumption of the model would be incorrect, which could account for the learning model's loss of accuracy in simulation season 2 players.

An alternative factor which may bias simulation accuracy may be habit. Players may build habits in season 1 which prove maladaptive in later seasons. If players were previously successful and did not

Username	*p-value*	*τ statistic*
aaaa	7.629×10^{-9}	0.962
apropos0	4.944×10^{-9}	0.969
basta	4.244×10^{-9}	0.934
Becccccca	2.711×10^{-9}	0.959
creilly1	1.232×10^{-7}	1.0
DavetheRave	5.239×10^{-9}	0.949
Deanerbeck	1.032×10^{-9}	0.974
DX13	8.022×10^{-11}	0.958
ECDr	2.035×10^{-7}	1.0
Ellen	6.935×10^{-10}	0.963
Ezzey	5.702×10^{-8}	0.995
Frp97	1.232×10^{-7}	1.0
Jhannah	3.117×10^{-8}	0.996
l17r	4.265×10^{-8}	1.0
Luca1802	3.930×10^{-9}	0.982
Martin	7.137×10^{-9}	0.943
Nari	6.937×10^{-9}	0.992
sstein	7.316×10^{-8}	0.999
timri	7.316×10^{-8}	0.999

Figure 7.9: Correlation for simulation using simple character pair distribution model on RPGLite Season 2

realise that they may need to adapt to a changed metagame, their gameplay may be biased. They may also have grown accustomed to a particular strategy when playing, and either regard an increased rate of lost games in the new season as a result of RPGLite's random nature or simply refuse to change their character pair preferences out of stubbornness. The learning model does not account for any of these biases, which may impact its effectiveness when applied to simulations of season 2. Further research is needed to demonstrate that more complex augmentations of models such as additions of behaviours and parameters can be reused across models without a loss of model accuracy; some suggestions are given in section 7.5.

Finally: advice implementing improvements to the model or instrumenting it to collect results have been successfully reused. If this were not so, the outcomes of games could not be recorded, and no players would be successfully modelled using the model of learning, as the confidence model would be unimplemented. As results are collected by instrumentation advice and user *ellen* is successfully simulated using the learning model in season 2, we infer that these pieces of advice have been successfully reused. Instrumentation for data collection is an example of logic typically included in model codebases but ancillary to their main concerns [GK99], making this an example of a cross-cutting concern in simulation & modelling which these experiments implement as aspects. These results add further evidence that aspects which augment models can be successfully reused, and so constitute portable modules in the context of simulation & modelling.

7.5 Discussion

The experiments in this chapter contribute an investigation into the viability of using aspect-oriented programming for simulation & modelling and its practical utility in terms of model accuracy and its viability as a technique.

The experiment in section 7.2 explores whether aspect-oriented changes to models can be realistic. It finds that altering minor properties of a model to improve its accuracy is feasible, and applies advice to change a distribution in RPGLite to produce player-specific gameplay simulations. This confirms that aspect-oriented tooling for simulation & modelling *can* be used to encode changes to models, but does not introduce changes on the scale of adding new behaviours or parameters to a model; it investigates the tooling's viability, but not its utility.

Section 7.3 contributes an investigation into the utility of aspect-oriented simulation & modelling by applying a model of novel behaviour with new parameters to the model of RPGLite. The model is accurate for 6 players out of 21. Considering additional factors such as or players' misunderstandings of the game or possible character selection criteria other than their success in games, this is a successful result. This model constitutes a study into the practicality of representing changes to a model as advice, and determines that advice can be used to amend models and improve their representation of a system under study.

Finally, section 7.4 presents an experiment which studies the modular nature of aspects representing changes to models. It is anticipated that aspects can be used to encode behaviour which cuts across different models; to investigate this, the experiments presented in section 7.2 and section 7.3 are run again against gameplay data collected from RPGLite's second season. The experiments show that some aspects *can* be applied to other models, in particular simple modifications to a model which do not require parameterisation or add additional behaviour. However, section 7.4 does not conclusively demonstrate this for advice which introduces new behaviours and parameters to a model. These findings may be biased by players' season 1 habits around character pair selection being carried into their season 2 behaviour. If players chose characters habitually rather than responding to RPGLite's changed metagame, no learning would be present in their gameplay data. Relatedly, players may not respond to the change of season with any behavioural change at all. This would mean that the learning model's assumption that players explore possible character pairs when they are inexperienced would not be appropriate: players would effectively begin the game with a high degree of initial confidence, which the model of learning does not represent. This would cause real-world gameplay data to compare less favourably to datasets produced by the simulation.

One player — *ellen* — was modelled accurately in season 2 in both experiments using the learning model. Also, the model applying a player-specific distribution of character pair selections to simulated players' choices produced successful results. Advice which instrumented the model or modified it to mitigate the impact of stalemates was also successfully reused in the experiments described in section 7.4. For these reasons, the fourth research question can be answered affirmatively — however, further research is needed to examine whether more complex changes to models suffer a loss of performance. This can be investigated by creating aspects which represent behavioural change found in many systems, such as the models of distraction implemented by Fuzzi-Moss[SW16] and used in prior research [WS18a] as discussed in chapter 3. Applying these aspects across different case studies may yield positive results. This research should take care to avoid factors biasing behaviours such as habits or mistaken assumptions.

Chapter 8

Future Work

The focus of this thesis is to develop a state-of-the-art aspect-oriented framework, to produce a suitable experimental environment to demonstrate its effectiveness, and to use that environment to investigate whether aspect orientation can be used to augment models with behavioural variance. is suitable for simulation purposes? It's about showing that we *can* use aspect orientation appropriately in simulation environments, and that aspect orientation can also lead us to realistic and nuanced simulations, too. Go back through the pdsf and lit review chapters to make this argument properly. . We have found that aspect orientation can be used to create realistic and nuanced simulations, have successfully produced a well-constrained environment to simulate for testing purposes, and have produced a novel aspect orientation framework which demonstrates novel and powerful weaving concepts; lots of opportunities for research outwith this thesis' scope present themselves.

This chapter describes some possibilities for the presented research to be extended in the future.

Add a RQ at the end of each research opportunity; this can and should be a contribution of the thesis, and the RQs would help to sell it as such.

8.1 Aspect-Oriented Metaprogramming in real-world Software Engineering

The combination of metaprogramming and aspect-orientation introduces new possibilities in aspect-orientation. In traditional aspect-oriented work, aspects treat their targets as black boxes; this leads to limitations which aspect-oriented metaprogramming can address.

Traditional aspects cannot intersperse their behavioural modifications with the work being done by their target, as they apply their logic before or after the target's execution. The "textbook" use case for aspect-orientation is logging: aspects can separate logging from the business logic they are applied to. However, a programmer in mainstream programming paradigms may wish to insert logging behaviour *within* their business logic rather than *around* it. Aspect-oriented metaprogramming enables this as the target can have logging logic woven within business logic, which remains decoupled within the codebase. To achieve this end goal without "within"-style aspect application would require a refactoring of business logic to create join points which traditional aspects could apply against; this breaks the "obliviousness" property of the aspect orientated philosophy discussed in section 2.1.1.

8.2 Augmentation of pre-existing models

Models which predict the future state of some system may be accurate in general, but cannot account for unforseeable events such as financial collapses, pandemics, or unpredictable weather events. Pre-existing models cannot account for these one-off shifts in system state due to their random nature. For example, models of world health over time could not account for the Covid19 pandemic, and models of local or global economies could not incorporate real-world data from the recession resulting from the pandemic. The 2008 financial crisis also impacted local and global economies, but could not be predicted before the event. The World3 model is an example of one which has provided accurate high-level predictions of the state of various global systems over many decades Find citations for World3 model , but predictions for recent years fail to account for these unforseeable events.

If the modification is represented as a special case within the model, a research software engineer must introduce their corrective code alongside the model's core logic, thereby tangling[1] the two. It

[1] For an overview on the tangling of cross-cutting concerns, see section 2.1.1.

therefore has the properties of a cross-cutting concern [Kic+97; FFN00], and can be factored into an aspect as such.

Research investigating the use of aspects to correct for one-off events could aim to show that an aspect can "nudge" a system's state in-line with real-world data when exceptional circumstances affect a modelled system but cannot be elegantly represented within the model's logic. A research question for this future work is therefore:

> *Can aspect orientation be used to introduce special cases to real-world systems to correct a model's predictions in when a system under research is altered by a freak event?*

lorum ipsum

"Freak event" might be too informal here but I'm not sure what to replace it with.

8.3 Aspect Orientation's utility for Research Software Engineers

A corollary of the research opportunities for aspect orientation's use in software engineering discussed in section 8.1 is that there is an opportunity for research software engineers to benefit from the adoption of aspect oriented programming. However, while aspect orientation's use in industrial software engineering has drawn criticism [Ste06; Prz10; CSS04][2] its use within research codebases is a special case where it may be more suitable. Related suggestions for the use of aspect orientation in research codebases were proposed by Gulyás and Kozsik (as discussed in section 2.2.1). However, the benefits proposed in earlier work concerned the design of the software itself. Aside from aspect-orientation's utility in software design in a research context, there are potential benefits for the *practice* of developing these codebases which may be of professional interest to research software engineers.

The resource constraints and stringent requirements for accuracy in research codebases present challenges which aspect-orientation could assist with. Compensating for a special case in a pre-existing model would require maintenance of the codebase, which takes time and could inadvertently alter its behaviour. Time is a scarce resource in research environments, and undesired changes to a model's behaviour can invalidate research results. An alternative to adjusting the models directly is to construct aspects which represent large and unpredictable events in real-world systems such as pandemics,

[2] A review of aspect orientation's critique is presented in section 2.1.7.

economic crises, war and famine. These can be modelled on real-world data for accuracy, which can produce realistic simulations as this thesis demonstrates. This use of aspect orientation in simulation and modelling can be investigated by creating a proof-of-concept of the approach as applied to pre-existing models.[3] Studies can also be conducted to investigate whether an aspectually-augmented model is quicker to construct and easier to maintain in future than a codebase with "patches" written into its original logic.

Researchers investigating this technique's application to existing models could also investigate the difficulties of augmenting a model created with no intention of weaving advice in the future. The construction of advice requires appropriate join points to be specified, and codebases which are structured in a way which doesn't yield convenient join points might be more complex to augment aspectually. These cases raise another use-case of PyDySoFu's "within"-style weaving through runtime metaprogramming: where other aspect orientation frameworks force aspects to treat the targets they are invoked on as black-boxes, PyDySoFu can make modifications within them. PyDySoFu's new kinds of aspects may therefore be more useful than those in other frameworks for researchers responsible for maintaining codebases written with no consideration for join-point specification, and would therefore be more universally applicable; this possibility requires future investigation.

If the research described successfully shows that aspectually-augmented simulations are easier and quicker for an RSE to maintain and deliver than direct maintenance of the model's codebase, then augmentation of existing models to improve their accuracy can follow in the community.

If I've got time, I'd love to push my codebase augmenting a World3 model to achieve just this with predictive implications for the 2008 financial crisis and the covid pandemic.

8.4 Hypothesising Possible System Dynamics via Aspects

Unpredictable events can cause discrepancies between simulated system state at a given time and the real-world system it models, as discussed in **??**. While this technique presents promising research opportunities, researchers face other kind of model uncertainty which aspect orientation could also counter.

In some scenarios, it is difficult or impossible to make predictions about a system's future states because its dynamics are actively being researched. Standard scientific practice is to create a model —

[3]This approach is similar to PyDySoFu's initial proof-of-concept study [WS18a] as discussed in chapter 3.

mathematical, computational, or otherwise — to create synthetic datasets which indicate accuracy if their predictions align with what is empirically observed. Cite popper, any point including studies that do this too? It's fairly table stakes as a concept... Some aspects of the system under study may be well-understood.

Rather than creating models of an existing system which encode its hypothesised behaviour, a naive model can be created which operates as the scientific consensus understands it. Hypothesised behaviour takes the form of aspects altering behaviour in any manner the hypothesis requires. Within-style aspects allow arbitrary modifications, to simulated behaviour, increasing the flexibility of the technique. Data produced by each model can be compared to empirically sourced data, and their similarity quantified, as demonstrated in chapter 6 and chapter 7. Our null hypothesis is that the naive model's similarity to empirically sourced data is greater than that of the aspectually augmented model; our experiment's hypothesis is that the behavioural change applied as aspects is more representative of the system under study than that of the community consensus.

This technique has a satisfying property: the hypothesis in a given experiment is clearly described by its aspectual representation, and if an experiment is successful — meaning its hypothesised behaviour accurately represents the real-world system — then the model adopted in future research as that of the community consensus can be the aspectually-augmented one produced by the original study. In this way, the scientific process is directly represented in the structure of the codebase, and the community's progression to increasingly accurate models of a system is represented by the progressive adoption of "patches" to an original theory.

Maybe too informal, but I don't know, maybe live a little...?

Hypotheses can also be created compositionally in this model. Researchers might develop a series of potential system properties or behaviours, but are unable to investigate all reasonable combinations in a timely manner. However, sets of aspects representing each can be composed to produce, for N hypothesised behaviours, 2^N potential behaviours which can be compared to empirical datasets to identify which combination of possible behaviours most closely resembles that of the real-world system under study. The technique is similar to modern pharmaceutical development, where a series of algorithmically identified compounds can be produced and tested to identify which are most suitable to treat a medical issue find a citation for modern algorithmic / robotic drug development , but can be applied to any models where aspectual augmentation is appropriate.

This sub-sec is already quite long, and including this

This technique for developing experimental model codebases has another desirable property: it unifies seemingly incompatible philosophies of the scientific process. Kuhn [Kuh12] explains the scientific process as inherently social: it starts with a paradigm which is accepted as broadly true, and accumulates an increasing number of exceptions until the paradigm itself is deemed unfit, and a new basis for a field's research is adopted. Popper [Pop13] explains the scientific process as an approximation towards truth, with incremental progress made with each result achieved by a research community. The proposed technique for developing experimental model codebases demonstrates features of both. Consider the original model a paradigm initially selected by community consensus, and the application of aspects Popper's incremental movements toward truth (as successful experiments are conducted) or Kuhn's exceptions to the agreed model (as experiments identify weaknesses in the original model). In this case, each successive new model adopted by a community is adopted in a Popplarian manner: improvements are objectively measured, incremental, and would trend towards truth as a model's behaviour fits empirical observations increasingly closely. However: over a sufficient period of time, the incremental patching of an original model would produce an accepted community model which contains a relatively large amount of discovery and complexity encoded in aspects, as compared to the original model they are applied to. One would expect the research community to rewrite the base model to simplify future aspect application and to more elegantly encode recent research findings; effectively discarding the original paradigm in favour of a new one. This process is Kuhn's "paradigm shift", where paradigms are dropped once a generation of researchers determine that an originally accepted theory on a topic is unfit for purpose as evidenced by mounting exceptions in the literature; a new paradigm is to be accepted by the community, as a new base model would have to be written and adopted.

The relevant philosophy of science is more nuanced than its brief explanation here, and the suitability of the approach for the development of research codebases is to be investigated; the work involved is outwith the scope of this thesis, but is suggested as future work. A basis for the incremental improvement of models via aspects is effectively demonstrated in chapter 6 and chapter 7, but the feasibility of the approach as a basis of a community's scientific process and relation to philosophy of science warrants further investigation.

Finally: hypothesised properties of a system might interact, meaning their discovery could involve the fitting of multiple models of possible behaviours to discover the properties of a real-world system. The use of aspects to encode — and discover the presence of — hypothesised behaviour is discussed in section 8.9.3 as the RPGLite dataset presents convenient opportunities for doing so. The optimisation of multiple aspectual models of behaviour is discussed in section 8.8.

8.5 Standards for Aspect Orientation in Research Codebases

The possibility of a research community sharing models containing aspectual augmentation and possibly developing additional aspects to augment a model further involves a considerable amount of model logic written within as advice. The community developing these aspects have the responsibilities of maintaining a codebase, but the added complexity that research software engineering introduces. These codebases may be used for many years, and may be iterated in in a series of future experiments. The legibility of these codebases and their long-term maintenance are areas of criticism in the software engineering community [Ste06; Prz10; CSS04]. The research community must therefore mitigate these weaknesses of the aspect-oriented paradigm when adopting the techniques discussed in this chapter for simulation and modelling.

To address the concern of the visibility of advice being woven, researchers may already take advantage of :

I've made this an enumerated list; is it better as a paragraph?

① Improvements to tooling produced by the aspect orientation research community, including IDE integration [CCK03] and runtime inspection [MR02], should make clear to engineers what advice is being woven in a codebase and assist with debugging aspect-oriented programs respectively.

② Aspect-orientation frameworks specifically designed to clarify to an engineer the aspects being woven should allow for less friction on the part of a maintainer who inherits a codebase and must reason about its behaviour. This is particularly important if the maintainer aims to weave more aspects into the codebase, and so must understand its existing behaviour before augmenting it further. Adopting weaving patterns such as import hook weaving — described in chapter 4 — should make a program clearer to a developer regardless of the tooling they have access to.

The impact of framework design on a codebase's maintenance should assist a developer even in the absence of tooling, but the success of import hook weaving in this regard is untested. An appropriate research question which arises is therefore:

> *Do aspect orientation frameworks with weaving techniques designed to simplify a developer's understanding of a program affect a codebase's long-term maintainability?*

8.6 Standard Aspect-Oriented Model Features

Researchers who build aspect-oriented models and extend others' aspect-oriented codebases must be able to collaborate at least as easily as they currently do in a culture without aspect orientation. One way aspect orientation might improve researchers' ease of collaboration is with standardised libraries for aspect construction. Similar libraries were developed when developing a case study for PyDySoFu's viability [WS18a; SW16], but development was left incomplete as discussed in section 3.3.1.

The original library with this aim, Fuzzi-Moss, was originally designed to provide standardised aspects to represent behavioural variance in socio-technical systems. The aspects developed in Fuzzi-Moss have no notion of being fitted to real-world data. Instead, they are simple models of behavioural variances such as distraction, which are parameterised to allow users of the library to use these simple models in whatever manner is appropriate for their use-case. A broader collection of these behavioural variations could simplify the use of aspect-oriented behavioural variation in the research community writ large, by removing researchers' burden to develop these themselves. Early construction of a library with Fuzzi-Moss' goals would also support researchers in sharing models or extending others', as they would be familiar with a common set of tools. The development of such a library and the production of case studies demonstrating its effectiveness would answer the research question:

> *Can researchers using aspect-oriented behavioural variance share a common set of tools to support and simplify the use of the technique in their codebases?*

Other tooling to support researchers in the development of aspectually augmented simulations and models could also be developed. For example, a library of fuzzers which make changes to an abstract syntax tree could be constructed. Such a library would not model specific behaviours, but would allow researchers to build models performing within-style aspect weaving without writing code which contained no metaprogramming logic. Instead, this logic would be encapsulated in utility functions provided by the proposed library. This would also reduce the work required of researchers looking to use the techniques demonstrated in this thesis. The library could also support the development of a Fuzzi-Moss-like set of socio-technical behavioural variances. However, such a library does not currently exist. A summary the contribution to the research community which the proposed library would make is:

> *The development of a library of metaprogramming operations which simplify the construction of within-style aspects, supporting its research use and demonstrated in case studies.*

The proposed library need not be a separate codebase to the aspect orientation framework it is designed alongside. PyDySoFu could contain this collection of metaprogramming operations; however, there is no obvious requirement for the library to be contained within PyDySoFu, and separating the proposed library from the framework it is built to interact with removes any dependancy on a particular aspect orientation framework; future researchers could develop alternative frameworks with within-style aspects which the proposed library could also be compatible with. We therefore suggest that the development of this library is a separate project from the development of PyDySoFu itself.

"obvious" is a little too informal, not sure what to replace it with here.

This might be a little rambly — consider omitting, rewording, or factoring into a footnote.

8.7 Testing Frameworks to detect Model Incorrectness

Many of the uses of aspect orientation in simulation & modelling research discussed in this chapter affect the construction and maintenance of a codebase, aspect orientation also has potential in the instrumentation of an experiment, as discussed in ??.[4] Experimental instrumentation was demonstrated for data collection in this thesis' experiments, as discussed in chapter 6 and chapter 7. We can get more precise than chapter references; find the sections / subsections. Instrumentation also has potential in other aspects of research software engineering; one possibility to be investigated is the use of aspects to alert researchers to impossible states visited by a model during runtime.

Many models construct a version of a real-world system which cannot achieve certain states. For

[4] In particular, see relevant notes on "The use of aspect-oriented programming in scientific simulations" [GK99].

example, a model of collisions might never be expected to see an increase in the total energy present across all modelled entities; simulations of socio-technical systems might expect bounds on the hours worked by a simulated workforce, or the amount of output the workforce creates; researchers might study a system to observe a n emergent property which is bounded by laws around its growth (for example, a linear increase in some property over time as opposed to exponential growth). It is important that any model of a real-world system accurately reflects the system's physical limits and underlying mechanics. Models may fail to reflect these mechanics for many reasons, including :

Are there any other reasons as to why models might be incorrect (and instrumentation could identify this by boundary checking)?

- The model may be developed with bugs which are not detected, i.e. it is conceptually incorrect;
- The model may be verified as correct but later maintenance introduces undetected errors, i.e. it contains bugs;
- The model may be correctly written, but given unexpected inputs.

Software engineering practices such as unit testing are common practice and may catch these bugs through identifying incorrect behaviour in components. Thorough test suites may also include integration tests which observe the behaviour of many model components working together. However, there are limitations of both testing techniques:

- Test suites as described may fail to identify non-deterministic errors in a model, which arise only in a small number of cases;
- Traditional test suites also may fail to identify emergent state errors, where individual components operate correctly but their repeated integration results in an incorrect emergent property of the system [5];
- Test suites usually fix values such as random seeds or input data, which may hide non-deterministic state errors or may not exhibit the properties of real-world data (and so exhibit different behaviour under test than when an experiment is conducted).

We propose the use of aspects to instrument models to assert a system state within expected bounds, both as components of a test suite and as assurance that a model was correct at any time it

[5]Emergent properties are often the state under study, as in PyDySoFu's original case study [WS18a]

is used as part of an experiment. All of these limitations can be alleviated by a test suite which is able assert the correctness of model state at different points in a program's execution. Models which non-deterministically reach erroneous states are primarily an issue during the execution of a model; aspectual state assertions need not alter a model's program flow, so are appropriate for use during its experimental use should any unexpected state arise. These states may also be deterministically caused by experimental input data which is not reflected by the inputs in a test suite; weaving these aspects during experimental use also addresses this limitation. Emergent properties of a system may also be tracked over time, and aspects may measure emergent properties to ensure they are within appropriate bounds at all times in an experiment and have correct properties of their own (for example, these aspects could ensure correct growth curves for the property in question).

Aspects can also be used as a regression test to detect the recurrence of bugs after they have been identified and fixed. If a simulated system enters an incorrect state during development, and the underlying bug is fixed, an aspect observing the system's erroneous property can be constructed which alerts developers if the bug is re-introduced.

Aspects instrumenting a model throughout its use in an experiment would also serve to guarantee peer reviewers of a study that the model was correct regarding the properties observed by the aspects. A suite of aspects which observe bounds on and characteristics of system properties is, functionally speaking, a declarative set of known-good behaviours of a model. Peer reviewers who sought to check that a model was correct should be able to extend this list of properties and observe any assertions during reproduction of a study's findings.

Within-style aspects are particularly appropriate for this task, as the measurement desired of a program might exist inside an existing function or block of code. It is conceivable that observing system state before and after a function executes would not be sufficiently granular to assert correct system state in some cases. Aspectual instrumentation of research codebases is therefore particularly useful in light of new weaving techniques in PyDySoFu, traditional aspect orientation frameworks would be sufficient in instrumenting at least some models for observation.

To demonstrate the efficacy of this approach, a library of tools to observe boundaries on system states could be developed, similarly to the library of tools suggested in section 8.6. Case studies augmenting research codebases using either this observational tooling or ordinary aspects can then be constructed. These should include new codebases, to demonstrate the technique's utility during development, but should also include the instrumentation of old models and the detection of any irregularities in existing, published work. Such a project would answer the research question:

> *Can aspect orientation be used to instrument a model and assert correctness by observing that the model's properties remain within expected bounds?*

8.8 Optimisation of Multiple Aspects

Experiments in chapter 6 and chapter 7 made use of models which were fitted to real-world data. However, they only fitted parameters of learning models, and only some parameters were fitted. This was partly due to the computational cost involved in fitting these parameters and time constraints on this research.

Is this point worthwhile? I cited it in IIRC an experimental chapter, but I'm unsure it's worth including. For future Tom to revisit.

Optimisation is an independent research field with a broad range of techniques outwith the scope of this thesis to explore; however, a study of optimisation algorithms and technologies as applied to the fitting of aspects to real-world system models is a topic for future research. Models of real-world systems can be computationally expensive to complete, meaning that measuring the accuracy of a model with a variety of parameters could be computationally expensive. A survey of optimisation techniques suitable for aspectually augmented simulations and models would answer the research question:

> *Which optimisation techniques are best suited to fitting parameters of aspects woven into models of real-world systems, with regards the accuracy of the aspectually augmented model produced?*

8.9 Future Work pertaining to RPGLite

RPGLite's dataset was analysed for the purposes it was collected for in this thesis: to aid in the realistic simulation of a well-controlled socio-technical system. However, some pieces of analysis have not been completed; this section discusses future work enabled by the dataset of real-world play which this thesis contributes.

8.9.1 Causes of Game Abandonment

The games included in the dataset produced by RPGLite finish in varying states: some are incomplete (as some games were ongoing when the snapshot of the game's database was taken), and others are finished. A third category of games are ones where a player opted to abandon a game they were participating in. This feature of RPGLite was developed to allow players who were playing inactive opponents to free a game slot. It also allowed players who felt they had no hope of winning, or were bored with a particular game to forfeit.

As many datapoints are included in the published dataset, it is possible to observe how active players were (by measuring the frequency with which they would open RPGLite), and engagement with the app might be quantified by players' exploration of (or frequent use of) leaderboards and customisation features. Correlations might exist between players' activity, engagement, or experience with their propensity to abandon a game in different states. For example, one hypothesis could be that particularly active players frequently forfeited games against inactive players out of boredom, or that players who were unlikely to win a game (or were inexperienced players of RPGLite paired against competent players) might be more likely to forfeit. These results might be of interest to the broader game design community. Research investigating reasons for abandonment would answer the research question:

> *Can players who are likely to forfeit a game be identified through patterns of RPGLite play?*

8.9.2 Patterns of Play within Cliques of Players

RPGLite's application implemented matchmaking systems which allowed players to advertise themselves as open to new games, challenge players discovered on a leaderboard, or rematch against players they had previously played against through a game history. However, two factors might have impacted how players chose their opponents:

① Recruiting methods means people might have known each other

② Players who find others they are well-matched against might repeatedly re-match rather than continuing to discover players they might have less satisfying games with

These factors would imply the formation of cliques of players: small groups who often created games against each other much more frequently than they did with the game's broader community. If so, this could have implications for uses of the PRGLite play dataset. For example, in the experiments discussed in chapter 6 and chapter 7 no structure is imposed on the matching of simulated players, so that players would be uniformly exposed to the game's community. As these experiments simulate learning from the outcomes of games, exposure to the entire community would expose players to learnings from all games played; cliques of players have a limited set of games to learn from, and so could affect what is learned by that clique. The broader community would also be insulated from what was learned within the clique, meaning that a loss of information affects all players.

Future research making use of the RPGLite dataset should be aware of any underlying patterns which could affect the dataset's emergent properties, such as learning styles. Clique formation is an example of such a pattern; others might yet present themselves. Future research making use of the dataset of RPGLite play may look to address research questions such as:

> *Are patterns such as cliques of players present in the RPGLite dataset? If so, what patterns exist, and what is likely to have driven the formation of those patterns?*

8.9.3 Aspect Oriented Discovery of Dataset Properties

Consider refactoring this into a subsection of that sec.

Relatedly, the dataset presents opportunities for research in aspectually augmented simulation & modelling related to the future work described in section 8.4. In section 8.4, the use of aspect orientation to encode hypothesised behaviour was discussed. However, in some scenarios, the expected behaviour may be unknown: it is currently unknown whether cliques were formed by RPGLite players; the behavioural impact of player interaction with ancillary game features such as leaderboards or profile customisation is unknown; players' early aptitude for RPGLite may cause them to lose interest relatively quickly due to a lack of challenge or spur them to climb leaderboards or continue to win games; players more likely to challenge others with more experience might learn and be more likely to continue playing RPGLite, or might quickly lose interest due to repeated loss.

These behaviours can be hypothesised, but many might have an impact on each other: players might find experienced peers through the leaderboard, so their progression in or engagement with the game could be due to their propensity to explore (by using the leaderboard) or their exposure to more advanced play (through games with experienced peers). This can be investigated through aspects fitted to real-world data, used to discover parameters for multiple hypothesised concerns in the player data by fitting them simultaneously.[6] The relative importance of each concern could be discovered by inferring parameters controlling the strength of each concern which were fitted to real-world data. This would allow the existance and relative impact of real-world behaviours to be discovered.

This proposal is an extension of the research discussed in section 8.4, but the RPGLite dataset is a suitable testing ground for the technique which already exists. Larger-scale data collection described later in this chapter (see) could provide opportunities to test this more rigorously, as the discovery of nuanced emergent properties of RPGLite play might require a considerably larger dataset than this thesis had the scope to contribute.

Research into this technique would contribute answers to the research question:

> *Can potentially interacting emergent properties in a real-world system by identified through the fitting of aspectual models representing hypothesised properties to empirical datasets which may exhibit them?*

8.9.4 Larger-scale Data Collection

RPGLite's playerbase was recruited informally. Only two seasons of gameplay were run, and the game was not heavily advertised to prospective players, particularly once it was clear that sufficient data would be collected for the research that required it. As a result, there is scope for a larger and longer-term data collection effort to be made. A re-release of the application in major mobile app stores

[6] Research into relevant techniques is proposed in section 8.8.

with a concerted effort to release new seasons of the game and maintain player interest for an extended period of time — perhaps with additional features, such as in-game chat, favourites lists of previous opponents, or match replay and analysis (with suggestions for improved play backed by the formal methods inherant in RPGLite's design) — would enable a richer analysis, and broader utility to the games research community.

RPGLite's dataset contains information about players' interactions with the application itself. As the game made available some features typical of modern games such as leaderboards, matchmaking, achievements, and graphical customisation, player data containing interaction data on features which are used commercially could be used to shed light on the more effective aspects of modern game design for engagement purposes. There are many opportunities available for the game design research community to investigate.[7] Publications in the field from co-creators of RPGLite reflect further on the design and future improvements of the game [**GameOn2020**; KM20; Kav21; Kav+19; KM21].

A large-scale user study of RPGLite could also be used in pursuit of other research questions suggested in this chapter. Players could be classified by their likelihood of abandoning games using the patterns identified in section 8.9.1 to verify the patterns empirically. A larger dataset might be useful in testing the discovery of system behaviours using hypothesised behaviours encoded in aspects. This could be used for both the aspectual encoding of an experimental setup, as discussed in section 8.4, the discovery of features of player behaviour suggested in section 8.9.3, the discovery of dataset properties such as player cliques including the real-time prediction of clique formation to during data collection for experimental verification of a clique classification process discussed in section 8.9.2, or the fitting of multiple aspects to models as described in section 8.8. Relevant research questions are discussed in their respective sections of this chapter.

8.10 Discussion

Possibly too grandiose? I don't think so really, but I wouldn't want to present it as being more than it is

The research presented in earlier chapters motivates a research effort spanning multiple fields, and has implications for experimental design as well as software engineering and simulation & modelling techniques. Earlier contributions add to the corpus of knowledge in their fields, but also carry

[7] Unfortunately, they remain outwith the scope of this thesis, which focuses on simulation technologies more than it does game design.

implications for our understanding of and research practice in those fields. This chapter contributes an investigation into these implications by exploring the possibilities these techniques make possible, and by explaining their potential impact.

In *The structure of scientific revolutions* [Kuh12], Kuhn notes:

> (…)new inventions of theory [are not] the only scientific events that have revolutionary impact upon the specialists in whose domain they occur. The commitments that govern normal science specify not only what sorts of entities the universe does contain, but also, by implication, those that it does not. It follows, though the point will require extended discussion, that a discovery like that of oxygen or X-rays does not simply add one more item to the population of the scientist's world. Ultimately it has that effect, but not until the professional community has re-evaluated traditional experimental procedures, altered its conception of entities with which it has long been familiar, and, in the process, shifted the network of theory through which it deals with the world. Scientific fact and theory are not categorically separable, except perhaps within a single tradition of normal-scientific practice. That is why the unexpected discovery is not simply factual in its import and why the scientist's world is qualitatively transformed as well as quantitatively enriched by fundamental novelties of either fact or theory.

(…)

The ideas presented in this thesis — in particular aspectually augmented simulations & models and aspect-oriented runtime metaprogramming — are not new "theories" in the relevant fields which contribute any profound or revolutionary new paradigms. However, they do present a new approach in simulation and modelling, and do so through new aspect orientation techniques. There are therefore contributions in both fields in which the "scientific fact and theory" Kuhn refers to are affected: in both fields new research opportunities and experimental practices are made feasible, with implications for both the research facts in the fields (for example, the contribution of the concept of aspect-oriented runtime metaprogramming, for example) and the theory of the fields (for example, the use of new aspect orientation technologies in experimental design).

New capabilities of aspect orientation frameworks are suggested, built, and demonstrated in experimental case studies through PyDySoFu. Their design and the extension of the tooling around these concepts present opportunities for future work as discussed in section 8.5 and in section 8.6, but the implications of their use in software engineering requires further study, as discussed in section 8.1, as well as in research software engineering specifically, as discussed in section 8.2, section 8.3, section 8.7 and section 8.8; further, the possibilities investigated in this research have implications for research in the practice of research itself, as discussed in section 8.4, section 8.7, section 8.8 and section 8.9.3. For this reason, the work presented in earlier chapters motivates further research in many areas. This thesis contributes not only the exploration of the ideas presented in earlier chapters, but a discussion of their implication for Kuhn's "scientific fact[s] and theor[ies]" in their relevant fields.

Maybe go into more detail on what was mentioned in specific cross-referenced sections here? A long list of sections this chapter contains without any extra explanation isn't very satisfying, even if it's useful within this framing of exploring what Kuhn meant and how it relates to this thesis.

The dataset produced by RPGLite also offers opportunities for further study. Research opportunities presented by the game design and formal methods are explored by Kavanagh [Kav21], but the dataset this thesis helped to contribute also offers opportunities in the mining of player behaviour and the dataset's use in future aspectually augmented modelling research, particularly if a large-scale data collection study is undertaken. These possibilities were discussed in section 8.9, in particular in section 8.9.1, section 8.9.2, and section 8.9.4.

Chapter 9

Closing Discussion

The discussion in this chapter summarises the findings made in earlier chapters, and provides some concluding remarks.

First, the contributions made in earlier chapters are summarised to provide an overview of the research as a whole. Next, some limitations of the findings and methodology are acknowledged in the spirit of scientific integrity. The chapter then ends by reviewing the core concept of the thesis: that parts of scientific models can be written as — or refactored to be — advice, using aspect-oriented programming. This concept is examined through the lens of earlier chapters' investigations & results.

9.1 Summary of Contributions

Several contributions are made in earlier chapters. This section summarises them individually to provide an overview of the presented research and its conclusions. These contributions were made in response to opportunities found in chapter 2's literature review, and are distinct from the contributions of earlier work as outlined in chapter 3.

A new aspect weaving technique aiming to improve aspect-oriented code's legibility The principle of obliviousness which forms part of aspect-oriented programming's design philosophy [FFN00; Kel08; CM06], but critics note that it makes the intended behaviour of a program difficult to ascertain, and should therefore be used in moderation [LC07] or not at all [Prz10; CSS04]. PyDySoFu applies aspect hooks to modules as they are imported, meaning that modules which *can* be altered by aspects are identified not only when weaving is applied, but when any module is used. As this design preserves obliviousness, function implementations still contain no reference to any potentially applied aspects — but any other module which invokes that function must import it with hooks applied.

Advice may still be woven anywhere within the module calling the function, but the entire module does not need to be searched for a call to PyDySoFu's weaver: developers can simply find the module's import to check whether it can be influenced by aspects. This reduces the possible parts of the codebase one must check to identify whether arbitrary changes could be made to a function's behaviour from anywhere at all to a single, well-defined point in the program, addressing the limitation of obliviousness raised by critics such as Leavens and Clifton [LC07] or Constantinides, Skotiniotis, and Stoerzer [CSS04] without sacrificing the design principle itself.

Development of a tool which is suitable for use in aspect-oriented modelling & simulation The application of advice to existing codebases requires some technique to overcome a lack of useful join points in a pre-existing model. In addition, aspect-oriented programming suffers a criticism that obliviousness limits program legibility, an important quality of software developed for research purposes. To address this, a redesign of PyDySoFu was developed which provided compatibility with modern versions of Python and improved on the design of its hook weaver to address criticisms of aspect-oriented programming's legibility by adding aspect hooks at import time, thus preserving obliviousness while improving legibility.

Demonstration that a model's behaviour can be augmented to accurately reflect a system under study using advice

This description list item label is too big and doesn't fit; think the environment needs some styling, using the enumitem package.

The review of relevant literature in chapter 2 observes that many research projects in aspect-oriented programming produce tooling, but do not confirm that tooling's hypothetical contribution empirically, and few case studies exist which demonstrate the benefits of aspect-oriented programming empirically. It is therefore particularly important that PyDySoFu is demonstrated to successfully represent changes to a model as advice.

PyDySoFu was used to apply advice which augmented a naive model of RPGLite play; character selection was altered to reflect the characters selected by real-world individuals, as observed in the RPGLite gameplay dataset [KWM20]. This advice successfully produced synthetic gameplay datasets which correlated with the individuals being modelled by the advice. Advice is therefore demonstrated to be a viable mechanism for the encoding of changes to a model, producing the expected datasets when applied. PyDySoFu is also shown to be used successfully in a simulation & modelling setting.

A model of RPGLite play, and corresponding dataset of player interactions Supporting the earlier contributions is a model of RPGLite play Cite repo for RPGLite model , paired with a dataset of RPGLite player interactions [KWM20]. Both are publicly available in support of other researchers' future work.

An exploration of the possibilities which aspect-oriented modelling yields The demonstration that one can successfully represent changes to models as aspects yields novel research opportunities which are largely undocumented in the literature. Having demonstrated that model behaviours can be woven as advice rather than calcifying the changes in a model's source code, a large body of future work can be produced investigating more specific uses of the technique and applying it in novel, ways.

The future work illustrated in chapter 8 follows the example of Marsh [Mar94a], whose thesis introduces a formalism of trust and also yields a large number of research opportunities. Marsh notes that there exists such a large number of avenues of research which existing literature does not identify in any fashion that their thorough discussion of now-feasible opportunities constitutes a contribution in its own right; "future work" in the context of their results doesn't only refer to improvements on their own findings, nor to more advanced versions of their own formalism, but to entirely new research projects across a variety of fields. Similarly, the application of aspect orientation to simulation & modelling supports novel research in applying the technique to existing codebases, using the technique to develop rigorous methodologies for the acceptance of changes to models, alternative forms of collaboration for research teams, and other "future work" possibilities. These possibilities are enumerated in chapter 8 as a thorough exploration of earlier contributions' significance. Where possible, all future work identified is accompanied by a specific research question, to qualify the particular contribution that work would yield and to simplify other researchers' engagement with the topic.

9.2 Limitations

Check with Tim: is it worth acknowledging some limitations in the conclusion? I know it's a little unconventional, but I think it also shows some awareness / humility. Hoping it makes questioning in the viva easier, because I can point to areas where I openly acknowledge what could be improved in the future; I'm following the scientific process, and aware of what "good" work looks like. At least, that's what I'm trying to demonstrate here.

As with all research, the contributions summarised in section 9.1 have some limitations. These are acknowledged here in the spirit of openness, and because identifying weaknesses in any piece of work helps to improve anything building on it.

PyDySoFu improves on problems with obliviousness, but does not entirely solve them A weakness of aspect-oriented programming which its critics identify [Ste06; CSS04; Prz10] is that aspect-orientation's principle of obliviousness makes a program more difficult to reason about. Obliviousness — that the join-point of some advice is unaware that a weaver might change it — complicates reasoning about a program, because it's unclear from reading its source code that additional logic may be included when it is run, where that logic is to be woven, and what it is to do. PyDySoFu's ability to weave changes *within* its join-point as opposed to before or after it introduces new ways for a program to exhibit unexpected logic. The only way to identify whether a program been affected by PyDySoFu is to identify how it is used, by observing where it is imported and whether PyDySoFu is utilised there. Other aspect-orientation frameworks provide tooling to facilitate easier inspection of aspect-oriented programs; at present, this is not provided for PyDySoFu.

Only one instance of aspect-oriented modelling is investigated The experiments described in chapter 6 and chapter 7 investigate the use of aspect-oriented programming in a simulation & modelling context, but only apply aspect orientation to one model, and specifically one which is socio-technical in nature. As advice is a feasible mechanism to represent changes to models, changes which do not relate to behavioural variance should be considered also. Promising related work includes research by Cieslak et al. [Cie+11] which represents details of plant growth in aspect-oriented L-systems. The technique could be used to model more socio-technical systems such as degraded modes [JS07] and epidemiology [OLe20] or tackling modelling problems in other fields, such as diverging behaviours in astronomical models [Kne+15; Sem+16] or cross-cutting concerns in business-process modelling [Cap+09; Sil+20] and botany [Cie+11]. Future work should prove aspect-oriented programming's utility in a broader array of contexts.

Further research is required into the portability of aspects across models Results for the third experiment, in section 7.3, investigated the portability of aspects across models and found mixed results. The model of learning appears to successfully model players in season 2 of RPGLite but is unable to model as many players as it could for season 1 regardless of how the model is used. One explanation of this result is that biased player and assumptions within the learning model impacted the aspect's suitability to modelling a second season of RPGLite. Aspects implementing the prior distribution model are successful in simulating players across seasons; therefore, the approach seems viable, but future research should clarify the weaknesses of the learning model as applied to season 2 of RPGLite.

9.3 Aspectually Augmented Models

This thesis presents research into the application of aspect-oriented programming to the development of models and simulations in a research setting. The use of aspects to represent changes to a model is novel. Core to the approach is the notion that parts of a model such as behavioural traits, additional parameters (and their impact on a model), and behaviour contingent on environmental factors can be suitably modularised from it as cross-cutting concerns. Similarly, minor changes to a model such as altering existing model properties can be represented as advice rather than as a change to the model's source, making the change easily enabled or disabled without adding complexity to the model itself.

Verification that the technique is viable — and a demonstration of it — required the development of new tooling and the application of that tooling to investigate the benefits augmenting a model using aspects. PyDySoFu was rewritten to support Python 3 and to weave aspect hooks at import time, allowing for dynamic and flexible weaving of advice. A system suitable for simulation — RPGLite — was designed, implemented as a mobile game, and released to collect data to support modelling efforts. A model of RPGLite was constructed, and a formalism of learning and confidence was defined, implemented, and used as the foundation of experiments investigating aspect-oriented modelling. Three experiments were designed and conducted to investigate whether aspect-oriented programming could be used to change models, to compose a model with more complex behaviours, and to build single units of model change which are applicable across multiple models. Positive results from these experiments indicate that the technique may apply to other fields and use cases, and the breadth of these opportunities was explored in its own chapter.

Aspect-oriented programming is shown to successfully modularise changes to models. Lots more can be done to explore how it can be used and to identify its limitations; the work presented shows that it is feasible and promising.

Appendix A

Details of Pattern Used when designing RPGLite's Naive Model

This appendix contains additional details on the actor, context, and environment variables used as arguments to the functions implementing RPGLite's workflow steps. Each simulation step receives these three arguments at a minimum. Because steps of the model are functions, and therefore valid join points, aspects applied to these have access to the entire state of the simulation.

Actor — allows the function to identify the actor performing the activity defined by the function. This argument is any object uniquely identifying an actor. While RPGLite's naive model did not use this variable to store actor-specific data, the object identifying the actor could be used to encapsulate actor-specific information across different workflows or different invocations of the same workflow.

Context — allows a workflow step to determine details of the current thread of work being undertaken by the actor. This is necessary because in some simulations, the same actor might pause and resume multiple occurrences of the same activity — for example, they might concurrently play three different matches in RPGLite. As a result, it is necessary to understand the context of the action being performed by the actor in question. This argument can be any object uniquely identifying the context of a piece of work, but should be mutable (such as a class or dictionary-like object) to permit the communication of information across invocations of different action-representing functions.

Environment — an actor's actions can be determined by the global environment they act within. There may be ancillary details to the actor's actions and the context of their particular thread of work which they are undertaken within which are used to determine behaviour, such as a landscape they traverse or other actors they might choose to interact with. An actor may also alter the state of that environment. Because all actors share access to a global environment, it can also function as an area of memory used for message passing or a space where actors can set values and flags to alter the environment of other actors.

This concept is related to environments in some other simulation frameworks, such as SimPy[dev21], where the environment controls scheduling and execution. However, other frameworks' environments impose model properties such as a model of time which may not be of interest to a researcher; this structure imposes no such constraints and its flexibility offers a general-case design pattern rather than the structure of a pre-existing framework, as it requires nothing more than a mutable function argument common throughout a model's implementation.

Each simulation step receives these three arguments at a minimum. Because steps of the model are functions, and therefore valid join points, aspects applied to these have access to the entire state of the simulation. This happens to be a useful property for aspect-oriented simulation & modelling, though not a necessary one.

Appendix B

Complete Description of Model Parameters

The `ModelParameters` class defines the parameters which generate data from an experiment. It is introduced in section 6.5.1, and its implementation is shown in fig. 6.10 and fig. 6.11. However, many parameters control the

starting confidence The initial value of the model's confidence curve.

assumed confidence plateau A corresponding high value for the model's confidence curve. As the curve's growth rate is calculated to align with the number of games completed by the real-world player being simulated, and the curve asymptotically approaches 1, a high value is required which represents the expected confidence of a real-world player having played a significant number of games.

curve inflection relative to numgames The proportion of games played by the real-world player being simulated at which the player's confidence curve should reach the parameter `assumed_confidence_plateau` . This allows for control over the growth rate of the curve while linking the rate of growth to the number of games played by the real-world player being simulated;

C The curve parameter of the confidence model's underlying birch curve;

prob bored The probability that a player with confidence above `assumed_confidence_plateau` becomes "bored" and stops playing RPGLite. These players are removed from the simulated playerbase and replaced with new players. This mechanism is discussed in further detail in section 6.5.4.

boredom enabled Whether the boredom mechanism discussed in section 6.5.4 is to be enabled.

training data The set of real-world games used as a training fold when optimising parameters for simulating a player. This is used when annealing toward optimal parameters for the learning model, and so is explained in more detail in section 7.1.

testing data The set of real-world games used as a testing fold when optimising parameters for simulating a player. This is used when annealing toward optimal parameters for the learning model, and so is explained in more detail in section 7.1.

iteration base The number of games to play when generating a synthetic dataset. If `boredom_enabled ==False`, the number of games is instead calculated to ensure that all synthetic players play the same number of games as were completed by the played being simulated, (calculated from the training or testing fold as appropriate).

number simulated players The size of the simulated playerbase.

advice A list of tuples defining advice to weave when generating data. Pieces of advice are uniquely defined by a tuple containing the type of aspect to weave (such as `"prelude"` or `"error_handler"`), a string-represented regular expression defining the join point of the advice, and a `callable` to use as an aspect when weaving the advice. As `ModelParameters` instances are serialised to disk to persist experiment results and Python functions are not seralisable, this may also be a string value; strings are IDs of aspects, and are replaced with their corresponding aspect on deserialisation.

players The player being simulated. This parameter is a list of player usernames to support simulating groups of players if required, though this functionality is not used in the experiments presented in chapter 7.

args Any additional arguments to pass when running a simulation.

kwargs Any additional keyword arguments to pass when running a simulation.

boredom period	The number of games completed by the simulated playerbase between checks of players' confidence and random selection for removal (default 25).
initial exploration	A value which controls the number of games individual players complete before their confidence & learning models control their character pair selection (default 28, to allow sufficient games to select every available pair exactly once).

Appendix C

Search for Significant Results

This algorithm searches for statistically significant results within p-value and τ correlation coefficient thresholds across a majority of folds from k-fold validation, as described in **??**.

```
def analyse_result_file(filepath:str, season:int=1, viability_budget:int=1) -> (str, list[Result], float, float):
    '''
    Analyses results in a file at `filepath` and …returns:
        - username
        - all params that are viable for all folds & against the total dataset
        - and the pval threshold at which we find those params
        - the stat threshold at which we find those params.
    viability_budget is the number of pval + stat combinations which yield significant results to return. In other words, when the budget is 1, The first pval + stat combo yielding statistically significant results will be the only one for which results will be returned; if the budget is 2, a second pair will be found before returning; if 3, another still; and so on. A budget of -1 indicates that all results should be returned.
    '''
    # Unpickle results and pop them into a dict, where keys are players and values are relevant results.
    all_results = list()
    with open(filepath, 'rb') as result_file:
        try:
            all_results = pickle.load(result_file)
        except EOFError:
            print(f"could not read [possibly empty?] result in file {filepath}")
            all_results = []
        except Exception as e:
            print("ERR!")
            print(e)
            raise e
```

```
…
    pvals = [0.01, 0.02, 0.035, 0.05]
    stats = [0.5, 0.4, 0.3, 0.2]
    pval_stat_index_map = [(pval_index, stat_index) for pval_index in range(len(pvals)) for stat_index in range(len(stats))]
    sorted_pval_stat_indices = sorted(pval_stat_index_map, key=sum)
    search_param_combos = list(map(lambda indices: (pvals[indices[0]], stats[indices[1]]), sorted_pval_stat_indices))

    username = filepath.split('/')[-1].split('-')[0]

    all_games = None

    for pval, stat in search_param_combos:
        print(username, pval, stat)
        within_threshold = list()
        combinations_seen = set()
        result = None
        globally_viable_params = list()
        params_to_mutate = None
        for fold in all_results:
            within_threshold_for_fold = list()
            for result in fold:
                if all_games is None:
                    all_games = result.params.training_data + result.params.testing_data
                if params_to_mutate is None:
                    params_to_mutate = copy(result.params)
                if result.within_acceptable_bounds(pval, stat):
                    within_threshold_for_fold.append((result.params.c, result.params.curve_inflection_relative_to_numgames, result.params.prob_bored))
                    combinations_seen.add((result.params.c, result.params.curve_inflection_relative_to_numgames, result.params.prob_bored))
            within_threshold.append(within_threshold_for_fold)

        commonality = dict() # c, rgr_coeff combo mapped to how many folds they appeared above threshold
        for combination in combinations_seen:
            commonality[combination] = reduce(lambda acc, fold_res: acc + (1 if combination in fold_res else 0), within_threshold, 0)
        ranked_params_for_player = sorted(commonality.items(), key=lambda x: x[1], reverse=True)
        passing_params_for_player = filter(lambda x: x[1]>=3, ranked_params_for_player)
```

```
...
        # Find a param set which passes more than it fails on testing
            folds, and also passes on the dataset as a whole.
        globally_viable_params = list()
        for possible_viable_param_set, _ in passing_params_for_player:
            params_to_mutate.c = possible_viable_param_set[0]
            params_to_mutate.curve_inflection_relative_to_numgames =
                possible_viable_param_set[1]
            params_to_mutate.prob_bored = possible_viable_param_set[2]
            params_to_mutate.boredom_enabled = params_to_mutate.
                prob_bored != 0.0001
            params_to_mutate.testing_data, params_to_mutate.training_data
                 = all_games, all_games
            test_against_all_player_games = params_to_mutate.
                run_experiment(testing=True, correlation_metric=kendalltau
                , season=season)
            if test_against_all_player_games.within_acceptable_bounds(
                pval, stat):
                globally_viable_params.append(
                    test_against_all_player_games)

        if len(globally_viable_params) > 0:
            viability_budget = viability_budget - 1
            if viability_budget == 0:
                break

    return username, globally_viable_params, pval, stat
```

Appendix D

Complete Result Table answering RQ2

TODO these tables fall off the end of the page. Reformat! (It's currently 1am, but these tables aren't quite fully baked yet, so revisit.)

Username	*fold #*	*p-value*	*τ statistic*
apropos0	1	0.9295560829453566	-0.013223592098145723
apropos0	2	0.543551087147816	-0.09089184624256738
apropos0	3	0.24967847077719418	0.1740513323160178
apropos0	4	0.9137449869231443	0.01604642463551945
apropos0	5	0.9647628029421551	0.006602422212782976
basta	1	0.1816194374918113	0.20462566640533308
basta	2	0.18248946612827355	0.2059628159428695
basta	3	0.07612323582082255	-0.272855015888954
basta	4	0.8577156891579766	-0.027709044747234814
basta	5	0.8661911974793254	0.026015295118650823
creilly1	1	0.35240946540386875	0.1408415780640769
creilly1	2	1.0	0.0
creilly1	3	0.12963600753361082	0.2286190426597633
creilly1	4	0.8772351814797427	-0.022972314182022596
creilly1	5	0.23450470832759351	0.17684896380472018
creilly2	1	0.7644615028216746	-0.04586638580669498
creilly2	2	0.3738292876597282	-0.13638531407402904
creilly2	3	0.17362238220159643	-0.21433767861360697
creilly2	4	0.328310729802638	0.14811336858631574
creilly2	5	0.9652366286655247	-0.0065855064974466625
cwallis	1	0.0415013132764738	0.3093048452693846
cwallis	2	0.3097043535187982	-0.15154965050057037
cwallis	3	0.9156021858411141	-0.01584867655605092
cwallis	4	0.9659578395732228	0.0063394706224203685
cwallis	5	0.689920709902269	0.06039860389867787
Deanerbeck	1	0.881413966884111	-0.023469523137789664
Deanerbeck	2	0.004650160066661944	-0.4401133171167231
Deanerbeck	3	0.9228899037702708	-0.01512874020122304
Deanerbeck	4	0.7933653799467553	0.04143965365710999
Deanerbeck	5	0.617370080862484	-0.07781714997961811
ECDr	1	0.36980925964418543	0.13489559954637306
ECDr	2	0.3439505882395243	-0.1413306545515153
ECDr	3	0.4344070895785651	0.11690221930822196
ECDr	4	0.5123199775611744	0.09847971421781773
ECDr	5	0.38565298773701395	0.12963938295936053
elennon	1	0.6831590303385897	0.06225988751474009
elennon	2	0.8578928320061399	-0.02749806661015982
elennon	3	0.7094791911120704	-0.056959703582473833
elennon	4	0.8770639630397704	0.023931519038060085
elennon	5	0.31841443329320684	-0.15169528069375557
Ellen	1	0.4173609577782533	-0.11952108728582653
Ellen	2	0.5641844970478596	0.08530901495354998
Ellen	3	0.13921559643746703	0.21826031618958391
Ellen	4	0.1642969752311254	0.20782616765906128
Ellen	5	0.5442947581369362	0.08956801917846717
Etess	1	0.291201011176352	0.16483767372268998
Etess	2	0.6942338351677287	-0.06105840084617214
Etess	3	0.431484384718381	0.1222120131676966
Etess	4	0.046612898050594605	-0.3075889702362141
Etess	5	0.5323480476649404	0.09594614036147774
Fbomb	1	0.16921401929250612	-0.2198877878822063
Fbomb	2	0.5803180551036174	0.08720827961883659
Fbomb	3	0.34901258267821544	0.14955808305450605
Fbomb	4	0.1405016420954379	0.23354024746948013
Fbomb	5	0.5525244548993048	0.09469610989712465
Frp97	1	0.45156575491741413	0.11887614636188644
Frp97	2	0.9523646984875704	-0.0094262053632344
Frp97	3	0.3865494332628323	0.13478934738386475
Frp97	4	0.9778079547585447	0.004378139362159142
Frp97	5	0.7253387704489176	-0.05492984459933149
georgedo	1	0.1851060401540079	0.20487048828935067
georgedo	2	0.07564136825095882	-0.27471958553261244
georgedo	3	0.8390968137012565	0.03170390147558708
georgedo	4	0.1203247115520158	0.24061753448870887

Username	*fold #*	*p-value*	*τ statistic*
apropos0	1	$6.069589430124279 \times 10^{-10}$	0.9642436861538818
apropos0	2	$8.079029120240886 \times 10^{-10}$	0.9698244380089833
apropos0	3	$1.1001934266917237 \times 10^{-9}$	0.982403317924857
apropos0	4	$7.194122983312375 \times 10^{-10}$	0.9883036912035246
apropos0	5	$1.6591682110243096 \times 10^{-9}$	0.9854310631217637
basta	1	$6.984416886646488 \times 10^{-9}$	0.9751373529625749
basta	2	$1.2543214139569811 \times 10^{-8}$	0.9534625892455924
basta	3	$1.219848016158136 \times 10^{-8}$	0.9637888196533972
basta	4	$7.626421141995402 \times 10^{-9}$	0.9523532664857335
basta	5	$1.1299509732985322 \times 10^{-8}$	0.9480678693136386
creilly1	1	$6.984424740718803 \times 10^{-10}$	0.9609023536933049
creilly1	2	$8.088739467118352 \times 10^{-10}$	0.9681333740581184
creilly1	3	$1.2922427728333273 \times 10^{-9}$	0.9702535449004767
creilly1	4	$1.2171496039225237 \times 10^{-9}$	0.9804286358922405
creilly1	5	$1.2922427728333273 \times 10^{-9}$	0.9702535449004767
creilly2	1	$1.1542973899562161 \times 10^{-8}$	0.9839131599805843
creilly2	2	$3.3346034737769985 \times 10^{-9}$	0.9865162369237952
creilly2	3	$2.243342049839241 \times 10^{-8}$	0.9630868246861536
creilly2	4	$6.0273326028474205 \times 10^{-9}$	0.9752492558885195
creilly2	5	$3.929678299475491 \times 10^{-9}$	0.9820613241770825
cwallis	1	$2.5139005283860464 \times 10^{-9}$	0.9704949588309458
cwallis	2	$1.9764148531238795 \times 10^{-9}$	0.9371259937730507
cwallis	3	$1.5580932730098084 \times 10^{-9}$	0.9683640522700839
cwallis	4	$1.8651136050223395 \times 10^{-9}$	0.9744805811819829
cwallis	5	$2.503434431199603 \times 10^{-9}$	0.9625334218796218
Deanerbeck	1	$4.741611516573964 \times 10^{-8}$	0.9793792286287206
Deanerbeck	2	$5.016702308111833 \times 10^{-8}$	0.9879271228182775
Deanerbeck	3	$2.323229622171393 \times 10^{-8}$	0.9819805060619656
Deanerbeck	4	$3.1170677281801914 \times 10^{-8}$	0.9959919678390986
Deanerbeck	5	$5.016702308111833 \times 10^{-8}$	0.9879271228182775
ECDr	1	$8.455080743459683 \times 10^{-10}$	0.9591663046625438
ECDr	2	$4.736270844907838 \times 10^{-9}$	0.9051392757105036
ECDr	3	$2.9140953345657096 \times 10^{-9}$	0.9302605094190632
ECDr	4	$1.1720693720109277 \times 10^{-9}$	0.9539392014169457
ECDr	5	$3.1098576834660653 \times 10^{-9}$	0.956948752938691
elennon	1	$3.96336950354485 \times 10^{-9}$	0.9727697526833043
elennon	2	$1.0470764761196705 \times 10^{-8}$	0.966091783079296
elennon	3	$1.3569408931049373 \times 10^{-8}$	0.933447121915197
elennon	4	$3.890138222908016 \times 10^{-9}$	0.9727697526833043
elennon	5	$7.626421141995402 \times 10^{-9}$	0.9523532664857335
Ellen	1	$2.5379129530364656 \times 10^{-9}$	0.9499679070317291
Ellen	2	$2.9693964876923854 \times 10^{-10}$	0.982423938239031
Ellen	3	$8.169101159404917 \times 10^{-10}$	0.9863408219546951
Ellen	4	$9.452655162863045 \times 10^{-10}$	0.9843740386976972
Ellen	5	$4.678753424418981 \times 10^{-10}$	0.9491726884870106
Etess	1	$1.1129691779367105 \times 10^{-8}$	0.9940297973880048
Etess	2	$2.4395953820300098 \times 10^{-8}$	0.9999999999999999
Etess	3	$1.3757770980180775 \times 10^{-8}$	0.9812063786211384
Etess	4	$2.582210122313199 \times 10^{-8}$	0.989743318610787
Etess	5	$9.369742817562114 \times 10^{-9}$	0.9780192938436515
Fbomb	1	$3.1170677281801914 \times 10^{-8}$	0.9959919678390986
Fbomb	2	$7.316467015672806 \times 10^{-8}$	0.9999999999999998
Fbomb	3	$1.231558254354619 \times 10^{-7}$	1.0
Fbomb	4	$5.7020695573988095 \times 10^{-8}$	0.9950859653474028
Fbomb	5	$5.7020695573988095 \times 10^{-8}$	0.9950859653474028
Frp97	1	$2.4395953820300098 \times 10^{-8}$	0.9999999999999999
Frp97	2	$3.9762073116976477 \times 10^{-8}$	0.9919677414109794
Frp97	3	$2.582210122313199 \times 10^{-8}$	0.989743318610787
Frp97	4	$4.741611516573964 \times 10^{-8}$	0.9793792286287206
Frp97	5	$4.741611516573964 \times 10^{-8}$	0.9793792286287206
georgedo	1	$4.718509821360782 \times 10^{-8}$	0.9697815168769666
georgedo	2	$7.629394231426926 \times 10^{-9}$	0.9620913858416693
georgedo	3	$9.369742817562114 \times 10^{-9}$	0.9780192938436515

Appendix E

Complete Result Table answering RQ3

Username	*p-value <*	*τ >*	*Confidence curve value*	*Confidence RGR modifier*	*Prob. bored*	*# folds*
apropos0	0.01	0.4	0.2	3	0.0625	3
apropos0	0.01	0.4	10	0.1	*disabled*	3
basta	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
creilly1	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
creilly2	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
cwallis	0.01	0.4	0.02	0.1	0.25	4
cwallis	0.01	0.4	0.1	0.3	0.25	4
cwallis	0.01	0.4	1	0.1	0.25	4
cwallis	0.01	0.4	10	0.1	0.25	4
cwallis	0.01	0.4	0.02	0.1	0.0625	3
cwallis	0.01	0.4	0.02	0.3	0.25	3
cwallis	0.01	0.4	0.02	1	0.015625	3
cwallis	0.01	0.4	0.02	1	0.0625	3
cwallis	0.01	0.4	0.02	1	1	3
cwallis	0.01	0.4	0.02	3	0.015625	3
cwallis	0.01	0.4	0.02	3	0.25	3
cwallis	0.01	0.4	0.1	0.1	*disabled*	3
cwallis	0.01	0.4	0.1	0.1	0.0625	3
cwallis	0.01	0.4	0.1	0.3	0.0625	3
cwallis	0.01	0.4	0.1	1	1	3
cwallis	0.01	0.4	0.2	3	0.25	3
cwallis	0.01	0.4	0.2	1	*disabled*	3
cwallis	0.01	0.4	0.2	1	1	3
cwallis	0.01	0.4	1	0.3	0.25	3
cwallis	0.01	0.4	1	3	*disabled*	3
cwallis	0.01	0.4	1	3	0.25	3
cwallis	0.01	0.4	5	3	0.015625	3
cwallis	0.01	0.4	5	3	0.0625	3
cwallis	0.01	0.4	10	1	0.0625	3
cwallis	0.01	0.4	10	1	0.25	3
cwallis	0.01	0.4	50	0.3	0.25	3
Deanerbeck	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
ECDr	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
elennon	0.035	0.3	50	0.1	0.0625	3
Ellen	0.01	0.3	10	0.3	0.015625	3
Etess	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
Fbomb	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
Frp97	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
georgedo	0.02	0.3	50	3	0.015625	3
georgedo	0.02	0.3	0.1	3	0.015625	3
georgedo	0.02	0.3	0.2	3	0.0625	3
georgedo	0.02	0.3	0.2	1	0.0625	3
georgedo	0.02	0.3	0.02	1	0.0625	3
l17r	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
Jamie	0.01	0.4	0.2	1	1	3
Jamie	0.01	0.4	1	3	0.25	3
Jamie	0.01	0.4	5	0.1	0.25	3
Jamie	0.01	0.4	0.02	1	0.015625	3
kubajj	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
Nari	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
Paddy	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
sstein	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
tanini	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
timri	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*

Figure E.1: Untruncated data from the experiment in section 7.3, generated using the aspect-oriented model of learning and applied to season 1.

Appendix F

Complete Result Table for RQ4 using Prior Distribution Model

Username	*p-value*	*τ statistic*
aaaa	7.629394231426926e-09	0.9620913858416693
aaaa	5.2911309830316467e-08	0.9479191551169758
aaaa	7.626421141995402e-09	0.9523532664857335
aaaa	7.071834710138214e-08	0.8842393310811687
aaaa	2.4230764145479525e-08	0.9125528002479737
apropos0	4.94386607807306e-09	0.9685125342119326
apropos0	2.362740616794553e-09	0.9607689228305228
apropos0	3.3601165170005016e-09	0.9843215373488934
apropos0	1.646556894454968e-09	0.9874734094849511
apropos0	2.909943990006779e-09	0.9865765724632495
basta	4.244305440694443e-09	0.9342963657194947
basta	2.0180763246386707e-09	0.9645739631583109
basta	1.9477073127466093e-09	0.9645739631583109
basta	2.6931680607997693e-09	0.9703784010607079
basta	2.795605711599785e-09	0.9499679070317291
Beccccca	2.711275692905026e-09	0.9588607403653148
Beccccca	5.797919688404042e-09	0.9564144928693535
Beccccca	9.266862632910002e-10	0.9663321941277055
Beccccca	9.054062499720839e-10	0.9574271077563379

Bibliography

[AB08] Bourouis Abdelhabib and Belattar Brahim. “JAPROSIM: a Java framework for process interaction discrete event simulation”. In: *Journal of Object Technology* 7.1 (2008), pp. 103–119.

[ABZ08] AU Aksu, Faruk Belet, and B Zdemir. “Developing aspects for a discrete event simulation system”. In: *Proceedings of the 3rd Turkish Aspect-Oriented Software Development Workshop*. Bilkent University. 2008, pp. 84–93.

[AGL98] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. “Mining process models from workflow logs”. In: *Advances in Database Technology—EDBT'98: 6th International Conference on Extending Database Technology Valencia, Spain, March 23–27, 1998 Proceedings 6*. Springer. 1998, pp. 467–483.

[BB14] Brahim Belattar and Abdelhabib Bourouis. “Yet Another Java Based Discrete-Event Simulation Library.” In: *J. Softw.* 9.1 (2014), pp. 82–88.

[Ben+16] Steve Benford et al. “On Lions, Impala, and Bigraphs: Modelling Interactions in Physical/Virtual Spaces”. In: *ACM Trans. Comput.-Hum. Interact.* 23.2 (May 2016). ISSN: 1073-0516. DOI: 10.1145/2882784. URL: https://doi.org/10.1145/2882784.

[BH02] Jason Baker and Wilson Hsieh. “Runtime aspect weaving through metaprogramming”. In: *Proceedings of the 1st international conference on Aspect-oriented software development - AOSD '02*. ACM Press, 2002. DOI: 10.1145/508386.508396. URL: https://doi.org/10.1145%2F508386.508396.

[Bir99] Colin PD Birch. “A new generalized logistic sigmoid growth equation compared with the Richards growth equation”. In: *Annals of botany* 83.6 (1999), pp. 713–723.

[BS11] Gordon Baxter and Ian Sommerville. "Socio-technical systems: From design methods to systems engineering". In: *Interacting with computers* 23.1 (2011), pp. 4–17.

[Bui14] JCAM Buijs. "Flexible evolutionary algorithms for mining structured process models". In: *Technische Universiteit Eindhoven* 57 (2014).

[Cal+14] Muffy Calder et al. "Real-time verification of wireless home networks using bigraphs with sharing". In: *Science of Computer Programming* 80 (2014), pp. 288–310.

[Cap+09] Claudia Cappelli et al. "An aspect-oriented approach to business process modeling". In: *Proceedings of the 15th workshop on Early aspects - EA '09*. ACM Press, 2009. DOI: 10.1145/1509825.1509828. URL: https://doi.org/10.1145%2F1509825.1509828.

[CBB13] Meriem Chibani, Brahim Belattar, and Abdelhabib Bourouis. "Toward an aspect-oriented simulation". In: *International Journal of New Computer Architectures and their Applications (IJNCAA)* 3.1 (2013), pp. 1–10.

[CBB14] Meriem Chibani, Brahim Belattar, and Abdelhabib Bourouis. "Practical benefits of aspect-oriented programming paradigm in discrete event simulation". In: *Modelling and Simulation in Engineering* 2014 (2014).

[CBB19] Meriem Chibani, Brahim Belattar, and Abdelhabib Bourouis. "Using aop in discrete event simulation: A case study with japrosim". In: *International Journal of Applied Mathematics, Computational Science and Systems Engineering* 1 (2019).

[CCK03] Andy Clement, Adrian Colyer, and Mik Kersten. "Aspect-oriented programming with AJDT". In: *ECOOP Workshop on Analysis of Aspect-Oriented Software*. Vol. 10. 2003.

[Cie+11] Mikolaj Cieslak et al. "Towards aspect-oriented functional–structural plant modelling". In: *Annals of Botany* 108.6 (July 2011), pp. 1025–1041. DOI: 10.1093/aob/mcr121. URL: https://doi.org/10.1093%2Faob%2Fmcr121.

[Cla73] R Lawrence Clark. "LINGUISTIC CONTRIBUTION TO GOTO-LESS PROGRAMMING". In: *Datamation* 19.12 (1973), pp. 62–63.

[CM06] Anis Charfi and Mira Mezini. "Aspect-Oriented Workflow Languages". In: *OTM Conferences*. 2006.

[CM07] Anis Charfi and Mira Mezini. "Ao4bpel: An aspect-oriented extension to bpel". In: *World wide web* 10.3 (2007), pp. 309–344.

[CMM10] Anis Charfi, Heiko Müller, and Mira Mezini. "Aspect-Oriented Business Process Modeling with AO4BPMN". In: *Modelling Foundations and Applications*. Ed. by Thomas Kühne et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 48–61. ISBN: 978-3-642-13595-8.

[Coh13] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. Academic press, 2013, pp. 79–80.

[con16] RafeKettler & github contributors. *magicmethods (A guide to Python's magic methods)*. https://github.com/RafeKettler/magicmethods/tree/65cc4a7bf4e72ba18df1ad17a377 2011 — 2016.

[con21] Clarete & github contributors. *ForbiddenFruit*. https://web.archive.org/web/20210515092416/https://github.com/clarete/forbiddenfruit. 2021.

[CS04] Ruzanna Chitchyan and Ian Sommerville. "Comparing dynamic AO systems". In: 2004.

[CSS04] Constantinos Constantinides, Therapon Skotiniotis, and Maximilian Stoerzer. "AOP Considered Harmful". In: *In Proceedings of European Interactive Workshop on Aspects in Software (EIWAS)*. 2004.

[DBM13] Jan De Houwer, Dermot Barnes-Holmes, and Agnes Moors. "What is learning? On the nature and merits of a functional definition of learning". In: *Psychonomic bulletin & review* 20 (2013), pp. 631–642.

[dev21] SimPy developers. *SimPy 4.0.1 documentation*. https://simpy.readthedocs.io/en/4.0.1/. 2021.

[DGM08] Adrian Dozsa, Tudor Girba, and Radu Marinescu. "How lisp systems look different". In: *2008 12th European Conference on Software Maintenance and Reengineering*. IEEE. 2008, pp. 223–232.

[Dij68] Edsger W Dijkstra. "Letters to the editor: go to statement considered harmful". In: *Communications of the ACM* 11.3 (1968), pp. 147–148.

[Dij82] Edsger W Dijkstra. "On the role of scientific thought". In: *Selected writings on computing: a personal perspective* (1982), pp. 60–66.

[Don20] Boudewijn van Dongen. "BPI Challenges: 10 years of real-life datasets". In: *IEEE Task Force on Process Mining Newsletter #2* (May 14, 2020). URL: https://www.tf-pm.org/newsletter/newsletter-stream-2-05-2020/bpi-challenges-10-years-of-real-life-datasets (visited on 05/14/2020).

[DOR95] DOV DORI. "Object-process Analysis: Maintaining the Balance Between System Structure and Behaviour". In: *Journal of Logic and Computation* 5.2 (Apr. 1995), pp. 227–249. ISSN: 0955-792X. DOI: 10.1093/logcom/5.2.227. eprint: https://academic.oup.com/logcom/article-pdf/5/2/227/6244720/5-2-227.pdf. URL: https://doi.org/10.1093/logcom/5.2.227.

[DR10] Robert Dyer and Hridesh Rajan. "Supporting dynamic aspect-oriented features". In: *ACM Transactions on Software Engineering and Methodology* 20.2 (Aug. 2010), pp. 1–34. DOI: 10.1145/1824760.1824764. URL: https://doi.org/10.1145%2F1824760.1824764.

[Esp04] Miklós Espák. "Aspect-Oriented Programming On Lisp". In: *International Conference on Applied Informatics*. 2004.

[FBS17] Daniel Friesel, Markus Buschhoff, and Olaf Spinczyk. "Annotations in Operating Systems with Custom AspectC++ Attributes". In: *Proceedings of the 9th Workshop on Programming Languages and Operating Systems*. 2017, pp. 36–42.

[FFN00] Robert E Filman, Daniel P Friedman, and Peter Norvig. "Aspect-oriented programming is quantification and obliviousness". In: (2000).

[FT+84] James France, John HM Thornley, et al. *Mathematical models in agriculture*. Butterworths, 1984.

[Gam+95] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0201633612.

[GK99] László Gulyás and Tamás Kozsik. "The use of aspect-oriented programming in scientific simulations". In: *Proceedings of Sixth Fenno-Ugric Symposium on Software Technology, Estonia*. 1999.

[GM96] Leonard Green and Joel Myerson. "Exponential versus hyperbolic discounting of delayed outcomes: Risk and waiting time". In: *American Zoologist* 36.4 (1996), pp. 496–505.

[Gol+08] Ryan M Golbeck et al. “Lightweight virtual machine support for AspectJ”. In: *Proceedings of the 7th international conference on Aspect-oriented software development*. 2008, pp. 180–190.

[Gom+95] Fabian Gomes et al. “SimKit: A High Performance Logical Process Simulation Class Library in C++”. In: *Proceedings of the 27th Conference on Winter Simulation*. WSC ’95. Arlington, Virginia, USA: IEEE Computer Society, 1995, pp. 706–713. ISBN: 0780330188. DOI: 10.1145/224401.224714. URL: https://doi.org/10.1145/224401.224714.

[Gom15] B. Gompertz. *On the Nature of the Function Expressive of the Law of Human Mortality, and on a New Mode of Determining the Value of Life Contingencies*. Jan. 1815. DOI: 10.1098/rspl.1815.0271. URL: https://doi.org/10.1098/rspl.1815.0271.

[GS04] Wasif Gilani and Olaf Spinczyk. “A family of aspect dynamic weavers”. In: *Dynamic Aspects Workshop (DAW04)*. 2004.

[GSK10] Ryan M Golbeck, Peter Selby, and Gregor Kiczales. “Late binding of AspectJ advice”. In: *Objects, Models, Components, Patterns: 48th International Conference, TOOLS 2010, Málaga, Spain, June 28–July 2, 2010. Proceedings 48*. Springer. 2010, pp. 173–191.

[GV82] Stephen Jay Gould and Elisabeth S Vrba. “Exaptation—a missing term in the science of form”. In: *Paleobiology* 8.1 (1982), pp. 4–15.

[Hem03] James F Hemphill. “Interpreting the magnitudes of correlation coefficients.” In: (2003).

[Hie+94] David Hiebeler et al. “The swarm simulation system and individual-based modeling”. In: Santa Fe Institute Santa Fe, NM, USA. 1994.

[Hil+00] Erik Hilsdale et al. “AspectJ: The Language and Support Tools”. In: *Addendum to the 2000 Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (Addendum)*. OOPSLA ’00. Minneapolis, Minnesota, USA: Association for Computing Machinery, 2000, p. 163. ISBN: 1581133073. DOI: 10.1145/367845.368070. URL: https://doi.org/10.1145/367845.368070.

[HJ09] Uwe D.C. Hohenstein and Michael C. Jäger. “Using Aspect-Orientation in Industrial Projects: Appreciated or Damned?” In: *Proceedings of the 8th ACM International Conference on Aspect-Oriented Software Development*. AOSD ’09. Charlottesville, Virginia, USA: Association for Computing Machinery, 2009, pp. 213–222. ISBN: 9781605584423. DOI: 10.1145/1509239.1509268. URL: https://doi.org/10.1145/1509239.1509268.

[HK11] Allan Hackshaw and Amy Kirkwood. "Interpreting and reporting clinical trials with results of borderline significance". In: *BMJ* 343 (2011). ISSN: 0959-8138. DOI: 10.1136/bmj.d3340. eprint: https://www.bmj.com/content/343/bmj.d3340.full.pdf. URL: https://www.bmj.com/content/343/bmj.d3340.

[How71] Nigel Howard. "Paradoxes of rationality: theory of metagames and political behavior". In: *(No Title)* (1971).

[HWL06] Erik Hollnagel, David D Woods, and Nancy Leveson. *Resilience engineering: Concepts and precepts*. Ashgate Publishing, Ltd., 2006.

[Ion+09] Tudor B Ionescu et al. "An Aspect-Oriented Approach for Disaster Prevention Simulation Workflows on Supercomputers, Clusters, and Grids". In: *2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*. IEEE. 2009, pp. 21–30.

[JP14] Paul Johannesson and Erik Perjons. *An introduction to design science*. Vol. 10. Springer, 2014.

[JS07] Chris W Johnson and Christine Shea. "A comparison of the role of degraded modes of operation in the causes of accidents in rail and air traffic management". In: *2007 2nd Institution of Engineering and Technology International Conference on System Safety*. IET. 2007, pp. 89–94.

[JWO12] Amin Jalali, Petia Wohed, and Chun Ouyang. "Aspect oriented business process modelling with precedence". In: *International Workshop on Business Process Modeling Notation*. Springer. 2012, pp. 23–37.

[Kav+19] William Kavanagh et al. "Balancing turn-based games with chained strategy generation". In: *IEEE Transactions on Games* 13.2 (2019), pp. 113–122.

[Kav21] William Kavanagh. "Using probabilistic model checking to balance games". PhD thesis. University of Glasgow, 2021.

[KDB91] Gregor Kiczales, Jim Des Rivieres, and Daniel G Bobrow. *The art of the metaobject protocol*. MIT press, 1991.

[Kel08] Stephen Kell. "A survey of practical software adaptation techniques." In: *J. Univers. Comput. Sci.* 14.13 (2008), pp. 2110–2157.

[Ken38] M. G. Kendall. "A New Measure of Rank Correlation". In: *Biometrika* 30.1/2 (1938), pp. 81–93. ISSN: 00063444. URL: http://www.jstor.org/stable/2332226 (visited on 12/16/2023).

[KH98] Ralph Keller and Urs Hölzle. "Binary component adaptation". In: *European Conference on Object-Oriented Programming*. Springer. 1998, pp. 307–329.

[Kic+01] G. Kiczales et al. "An Overview of AspectJ". In: *ECOOP*. 2001.

[Kic+97] Gregor Kiczales et al. "Aspect-oriented programming". In: *European conference on object-oriented programming*. Springer. 1997, pp. 220–242.

[KM20] William Kavanagh and Alice Miller. "Gameplay Analysis of Multiplayer Games with Verified Action-Costs". In: *The Computer Games Journal* 10.1-4 (Dec. 2020), pp. 89–110. DOI: 10.1007/s40869-020-00121-5. URL: https://doi.org/10.1007%2Fs40869-020-00121-5.

[KM21] William Kavanagh and Alice Miller. "Gameplay analysis of multiplayer games with verified action-costs". In: *The Computer Games Journal* 10 (2021), pp. 89–110.

[Kne+15] Alexander Knebe et al. "nIFTy cosmology: comparison of galaxy formation models". In: *Monthly Notices of the Royal Astronomical Society* 451.4 (June 2015), pp. 4029–4059. ISSN: 0035-8711. DOI: 10.1093/mnras/stv1149. eprint: https://academic.oup.com/mnras/article-pdf/451/4/4029/3857619/stv1149.pdf. URL: https://doi.org/10.1093/mnras/stv1149.

[Kok+21] Athanasios Kokkinakis et al. "Metagaming and metagames in Esports". In: *International Journal of Esports* (2021).

[Kuh12] Thomas S Kuhn. *The structure of scientific revolutions*. University of Chicago press, 2012.

[KWM20] William Kavanagh, Tom Wallis, and Alice Miller. "RPGLite player data and lookup tables". In: (2020).

[Lac97] Sheldon J Lachman. "Learning is a process: Toward an improved definition of learning". In: *The Journal of psychology* 131.5 (1997), pp. 477–480.

[LC07] Gary T Leavens and Curtis Clifton. "Multiple concerns in aspect-oriented language design: a language engineering approach to balancing benefits, with examples". In: *Proceedings of the 5th workshop on Software engineering properties of languages and aspect technologies*. 2007, 6–es.

[Li10] Chen Li. "Mining process model variants: Challenges, techniques, examples". PhD thesis. University of Twente, The Netherlands, 2010.

[Lin68] Aristid Lindenmayer. "Mathematical models for cellular interactions in development I. Filaments with one-sided inputs". In: *Journal of theoretical biology* 18.3 (1968), pp. 280–299.

[LSS09] Russell Lock, Tim Storer, and Ian Sommerville. "Responsibility modelling for risk analysis". In: (2009).

[Mac+11] Idarlan Machado et al. "Managing variability in business processes". In: *Proceedings of the 2011 international workshop on Early aspects - EA '11*. ACM Press, 2011. DOI: 10.1145/1960502.1960508. URL: https://doi.org/10.1145%2F1960502.1960508.

[MAG12] Giselle Machado, Vander Alves, and Rohit Gheyi. "Formal Specification and Verification of Well-formedness in Business Process Product Lines". In: *6th Latin American Workshop no Aspect-Oriented Software Development: Advanced Modularization Techniques, LAWASP*. Vol. 12. 2012.

[Mar03] Robert Cecil Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[Mar94a] Stephen Paul Marsh. "Formalising trust as a computational concept". In: (1994).

[Mar94b] Robert Martin. "OO design quality metrics". In: *An analysis of dependencies* 12.1 (1994), pp. 151–170.

[Mat08] Norm Matloff. "Introduction to discrete-event simulation and the simpy language". In: *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August* 2.2009 (2008), pp. 1–33.

[MD11] Richard Millham and Evans Dogbe. "Aspect-Oriented Security and Exception Handling within an Object Oriented System". In: *2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*. July 2011, pp. 321–326. DOI: 10.1109/COMPSACW.2011.60.

[Mil08] Robin Milner. "Bigraphs and their algebra". In: *Electronic Notes in Theoretical Computer Science* 209 (2008), pp. 5–19.

[Mil09] Robin Milner. *The space and motion of communicating agents*. Cambridge University Press, 2009.

[Mit+17] Alexey A. Mitsyuk et al. "Generating event logs for high-level process models". In: *Simulation Modelling Practice and Theory* 74 (May 2017), pp. 1–16. DOI: 10.1016/j.simpat.2017.01.003. URL: https://doi.org/10.1016%2Fj.simpat.2017.01.003.

[MK03] Hidehiko Masuhara and Gregor Kiczales. "A modeling framework for aspect-oriented mechanisms". In: *Proceeding ECOOP'03*. 2003.

[MR02] Katharina Mehner and Awais Rashid. "Towards a standard interface for runtime inspection in AOP environments". In: *Workshop on Tools for Aspect-Oriented Software Development at OOPSLA*. Vol. 2. 2002.

[MS16] A. A. Mitsyuk and I. S. Shugurov. "On process model synthesis based on event logs with noise". In: *Automatic Control and Computer Sciences* 50.7 (Dec. 2016), pp. 460–470. DOI: 10.3103/s0146411616070154. URL: https://doi.org/10.3103%2Fs0146411616070154.

[Nie77] Jurg Nievergelt. "Information Content of Chess Positions". In: *SIGART Bull.* 62 (Apr. 1977), pp. 13–15. ISSN: 0163-5719. DOI: 10.1145/1045398.1045400. URL: https://doi.org/10.1145/1045398.1045400.

[Obj11] Object Management Group (OMG). *Business Process Model and Notation, Version 2.0*. OMG Document Number formal/2011-01-03 (https://www.omg.org/spec/BPMN/2.0). 2011.

[Obj15] Object Management Group (OMG). *Unified Modelling Langauge, Version 2.5*. OMG Document Number formal/2015-03-01 (http://www.omg.org/spec/UML/2.5). 2015.

[OLe20] Aran O'Leary. "Simulating Human Behaviour in a Micro-epidemic using an Agent-based Model with Fuzzing Aspects". In: *Unpublished masters thesis* (2020).

[PAG03] Andrei Popovici, Gustavo Alonso, and Thomas Gross. "Just-in-Time Aspects: Efficient Dynamic Weaving for Java". In: *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*. Association for Computing Machinery, 2003, pp. 100–109.

[Par72] D. L. Parnas. "On the criteria to be used in decomposing systems into modules". In: *Communications of the ACM* 15.12 (Dec. 1972), pp. 1053–1058. DOI: 10.1145/361598.361623. URL: https://doi.org/10.1145%2F361598.361623.

[Pas+19] William Pasmore et al. "Reflections: sociotechnical systems design and organization change". In: *Journal of Change Management* 19.2 (2019), pp. 67–85.

[PGA02] Andrei Popovici, Thomas Gross, and Gustavo Alonso. "Dynamic Weaving for Aspect-Oriented Programming". In: *Proceedings of the 1st International Conference on Aspect-Oriented Software Development*. AOSD '02. Enschede, The Netherlands: Association for Computing Machinery, 2002, pp. 141–147. ISBN: 158113469X. DOI: 10.1145/508386.508404. URL: https://doi.org/10.1145/508386.508404.

[Pop13] Karl Popper. "The Logic and Evolution of Scientific Theory". In: Routledge, 2013, pp. 3–22.

[Pou+15] Asef Pourmasoumi et al. "On Business Process Variants Generation." In: *CAiSE Forum*. 2015, pp. 179–188.

[Prz10] Adam Przybylek. "What is Wrong with AOP?" In: *ICSOFT (2)*. Citeseer. 2010, pp. 125–130.

[Prz18] Adam Przybyłek. "An empirical study on the impact of AspectJ on software evolvability". In: *Empirical Software Engineering* 23.4 (2018), pp. 2018–2050.

[Raj+06] Hridesh Rajan et al. *Nu: Towards an AspectOriented Invocation Mechanism*. Tech. rep. Technical Report 414, Iowa State University, Department of Computer Science, 2006.

[Ric59] F. J. Richards. "A Flexible Growth Function for Empirical Use". In: *Journal of Experimental Botany* 10.2 (June 1959), pp. 290–301. ISSN: 0022-0957. DOI: 10.1093/jxb/10.2.290. eprint: https://academic.oup.com/jxb/article-pdf/10/2/290/1411755/10-2-290.pdf. URL: https://doi.org/10.1093/jxb/10.2.290.

[Ros97] Jarrett Rosenberg. "Some misconceptions about lines of code". In: *Proceedings fourth international software metrics symposium*. IEEE. 1997, pp. 137–142.

[RTL09] Payam Refaeilzadeh, Lei Tang, and Huan Liu. "Cross-Validation". In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 532–538. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_565. URL: https://doi.org/10.1007/978-0-387-39940-9_565.

[SA13] Thomas Stocker and Rafael Accorsi. "Secsy: Security-aware synthesis of process event logs". In: *Proceedings of the 5th International Workshop on Enterprise Modelling and Information Systems Architectures, St. Gallen, Switzerland*. Citeseer. 2013.

[SA14] Thomas Stocker and Rafael Accorsi. "SecSy: A Security-oriented Tool for Synthesizing Process Event Logs." In: *BPM (Demos)*. 2014, p. 71.

[Sch+14] Chris Schaefer et al. "Introducing Spring AOP". In: *Pro Spring: Fourth Edition* (2014), pp. 161–239.

[Sem+16] Federico Sembolini et al. "nIFTy galaxy cluster simulations – II. Radiative models". In: *Monthly Notices of the Royal Astronomical Society* 459.3 (Apr. 2016), pp. 2973–2991. ISSN: 0035-8711. DOI: 10.1093/mnras/stw800. eprint: https://academic.oup.com/mnras/article-pdf/459/3/2973/8107345/stw800.pdf. URL: https://doi.org/10.1093/mnras/stw800.

[Seo11] Simon Seow. "OBASHI White Paper". In: *APMG-International, High Wycombe, UK* (2011).

[Sil+20] Hercules Sant Ana da Silva Jose et al. "Implementation of Aspect-oriented Business Process Models with Web Services." In: *Bus. Inf. Syst. Eng.* 62.6 (2020), pp. 561–584.

[SL07] Olaf Spinczyk and Daniel Lohmann. "The design and implementation of AspectC++". In: *Knowledge-Based Systems* 20.7 (2007), pp. 636–651.

[SM14] Ivan Shugurov and Alexey A Mitsyuk. "Generation of a set of event logs with noise". In: *Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering*. 8. [illegible] … 2014.

[Spe04] C. Spearman. "The Proof and Measurement of Association between Two Things". In: *The American Journal of Psychology* 15.1 (1904), pp. 72–101. ISSN: 00029556. URL: http://www.jstor.org/stable/1412159 (visited on 12/16/2023).

[Ste06] Friedrich Steimann. "The paradoxical success of aspect-oriented programming". In: *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications - OOPSLA '06*. ACM Press, 2006. DOI: 10.1145/1167473.1167514. URL: https://doi.org/10.1145%2F1167473.1167514.

[Sto23] Tim Storer. *Theatre_AG*. https://web.archive.org/web/20230728153002/https://github.com/twsswt/theatre_ag. 2023.

[Sul01] Gregory T Sullivan. “Aspect-oriented programming using reflection and metaobject protocols”. In: *Communications of the ACM* 44.10 (2001), pp. 95–97.

[SW16] Tim Storer and William Wallis. *Fuzzi-Moss*. https://web.archive.org/web/20201018121800/https://github.com/twsswt/fuzzi-moss. 2016.

[TB51] Eric Lansdown Trist and Kenneth W Bamforth. “Some social and psychological consequences of the longwall method of coal-getting: An examination of the psychological situation and defences of a work group in relation to the social structure and technological content of the work system”. In: *Human relations* 4.1 (1951), pp. 3–38.

[Tha+17] Dai Hai Ton That et al. “Sciunits: Reusable Research Objects”. In: *2017 IEEE 13th International Conference on e-Science (e-Science)*. IEEE, Oct. 2017. DOI: 10.1109/escience.2017.51. URL: https://doi.org/10.1109%2Fescience.2017.51.

[Van+05] Boudewijn F Van Dongen et al. “The ProM framework: A new era in process mining tool support”. In: *Applications and Theory of Petri Nets 2005: 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005. Proceedings 26*. Springer. 2005, pp. 444–454.

[VD09] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[Ver45] PF Verhulst. “La loi d’accroissement de la population”. In: *Nouveaux Memories de l’Académie Royale des Sciences et Belles-Lettres de Bruxelles* 18 (1845), pp. 14–54.

[Von38] Ludwig Von Bertalanffy. “A quantitative theory of organic growth (inquiries on growth laws. II)”. In: *Human biology* 10.2 (1938), pp. 181–213.

[VVK09] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. “The refined process structure tree”. In: *Data & Knowledge Engineering* 68.9 (2009), pp. 793–818.

[VW04] Wil MP Van der Aalst and Anton JMM Weijters. “Process mining: a research agenda”. In: *Computers in industry* 53.3 (2004), pp. 231–244.

[Wal+21] William Wallis et al. "Designing a mobile game to generate player data - lessons learned THIS IS A DUPLICATE FIXME THIS IS A DUPLICATE FIXMETHIS IS A DUPLICATE FIXMETHIS IS A DUPLICATE FIXME THIS IS A DUPLICATE FIXME THIS IS A DUPLICATE FIXME THIS IS A DUPLICATE FIXME Duplicate citation to clean up ". In: *CoRR* abs/2101.07144 (2021). arXiv: 2101.07144. URL: https://arxiv.org/abs/2101.07144.

[WC10] Paul Wallis and Fergus Cloughley. *The OBASHI Methodology*. The Stationery Office, 2010.

[WJ97] AR Werker and KW Jaggard. "Modelling asymmetrical growth curves that rise and then fall: Applications to foliage dynamics of sugar beet (Beta vulgarisL.)" In: *Annals of Botany* 79.6 (1997), pp. 657–665.

[WS18a] Tom Wallis and Tim Storer. "Modelling realistic user behaviour in information systems simulations as fuzzing aspects". In: *International Conference on Advanced Information Systems Engineering*. Springer. 2018, pp. 254–268.

[WS18b] Tom Wallis and Tim Storer. "Process Fuzzing as an Approach to Genetic Programming". In: *Proceedings of the SICSA Workshop on Reasoning, Learning and Explainability*. Ed. by Kyle Martin, Nirmalie Wiratunga, and Leslie S. Smith. Vol. 2151. CEUR Workshop Proceedings. Aberdeen, UK: CEUR-WS.org, June 2018. URL: http://ceur-ws.org/Vol-2151/Paper_S3.pdf.

[WS18c] Tom Wallis and Tim Storer. *PyDySoFu*. https://github.com/twsswt/pydysofu. 2018.

[WS23] Tom Wallis and Tim Storer. *ASP*. https://web.archive.org/web/20230730164214/https://github.com/probablytom/asp. 2023.

[YC79] Edward Yourdon and Larry L Constantine. "Structured design. Fundamentals of a discipline of computer program and systems design". In: *Englewood Cliffs: Yourdon Press* (1979).