FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Applying Information Theory to Formal Models of Play

Exploratory Modelling of User Interaction in Real Time

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Simon Wallner BSc
Matrikelnummer 0625104

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn Michael Wimmer
Mitwirkung: Michael Hecher MSc

Wien, 29. September 2015

Simon Wallner	Michael Wimmer

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Applying Information Theory to Formal Models of Play

Exploratory Modelling of User Interaction in Real Time

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media Informatics and Visual Computing

by

Simon Wallner BSc
Registration Number 0625104

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn Michael Wimmer
Assistance: Michael Hecher MSc

Vienna, 29th September, 2015

Simon Wallner	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Simon Wallner BSc
Herzgasse 2A/12, 1100 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 29. September 2015

Simon Wallner

Acknowledgements

I like to thank my thesis supervisors Michael Wimmer and Michael Hecher for their patient and enduring support, especially over the last few months. It probably was not always that easy to work with me and to convince me of my errors and mistakes, but they gave me the opportunity to learn a lot in the process. I further want to thank Matthias Bernhard who was acting as a supervisor in very early stages of this work.

The cooperation on routinisation in games with Martin Pichlmair gave this work another important push and even more relevance. I would also like to thank the designers, developers and professionals that provided valuable insight into the playtesting practice through the interviews conducted in this thesis.

Lastly I want to thank my family, friends and my fellow colleagues for their kind support and for listening to my lamentation when things were not running as they should. Special thanks goes to Broken Rules for granting me access to their game and for providing additional data. Special thanks also to Vincent Wagner for generating hours of test data that unfortunately never made it into the final thesis.

Kurzfassung

Diese Diplomarbeit entwirft ein formelles Modell von Interaktion in Spielen, das als Werkzeug zur Analyse und zum Testen von Spielen eingesetzt werden kann. Im Modell wird Interaktion durch das Analysieren von grundsätzlichen Strukturen von Game-Controller Eingaben quantifiziert. Diese Eingaben werden durch Markov Ketten modelliert und Informationstheorie wird angewandt um den Unterschied zwischen dem Modell und der tatsächlichen Eingabe zu messen.

Das Modell verwendet spielunabhängige Game-Controller-Eingaben, die den kleinsten gemeinsamen Nenner für eine große Gruppe von Spielen darstellen. Die Modelle werden dynamisch zur Laufzeit für die individuellen Spielesessions trainiert. Dadurch können individuelle Analysen der SpielerInneninteraktionen durchgeführt werden, ohne aber die Methode selbst speziell auf einzelne Spiele anpassen zu müssen.

Um sich schnell an neue Spielsituation anpassen zu können, basieren die Modelle auf Daten der letzten paar Sekunden und Minuten. Dadurch kann das Problem entstehen, dass nicht genügen Daten zur Verfügung stehen um alle Modellparameter schätzen zu können. Dieses Problem wird dadurch gelöst, dass die vollen Wahrscheinlichkeitsverteilungen der einzelnen Parameter in Betracht gezogen werden.

Durch die Modellierung von Game-Controller-Eingaben und die Quantifizierung der Ergebnisse durch Informationstheorie leistet diese Arbeit einen Beitrag zum besseren Verständnis von Interaktion in Spielen. Das beschriebene Modell wurde in Software umgesetzt und vorläufige Ergebnisse einer ersten Vorstudie wurden gesammelt.

In dieser exploratorischen Studie zeigte die Analyse von neun Spielen unterschiedlicher Genres unterschiedliche Interaktionsmuster. Eines dieser Muster ist Routinisierung, ein Prozess bei dem eine Aktion wiederholt durchgeführt wird, bis sie zur Routine wird. Fortführende Untersuchungen in diesem Gebiet wurden in Kooperation mit Martin Pichlmair durchgeführt, und werden als works-in-progress Artikel im Tagungsband der ACM CHIPLAY Konferenz veröffentlicht werden [Wallner et al., 2015].

Abstract

This thesis proposes a formal model of interaction in games, to be used as tool for game analysis and game testing. The model allows a quantification of interaction by looking at the low-level structure and patterns in game-controller input. The game-controller input is modelled using discrete-time, discrete-space Markov chains, and information theory is used to quantify the mismatch between the model's prediction and the actual user input.

The model uses game-agnostic game controller data as its input, which is the lowest common denominator for a large class of games (almost all game console games, most PC games). The models are trained dynamically on-the-fly for each individual play session. This allows performing individual analyses of players' interactions, while still retaining an approach that is very general and can be used with different games without modification.

To adapt to new play situations quickly, the used models are only based on data from the last couple of seconds or minutes. This can lead to the problem that not enough samples may be available to confidently estimate all dynamic model parameters. This problem is mitigated by considering the full probability distribution of each parameter instead, using a beta distribution.

This work contributes to the understanding of interaction in games, modelling of raw user input and quantifying the model output using information theory. The described approach has been implemented in software and preliminary results from a prestudy are available.

In this exploratory prestudy, the post hoc analysis of nine different games from various genres revealed a number of interaction patterns. One of the observed patterns is routinization, a process in which an action is performed repeatedly until it is executed almost unconsciously. Research in this field, based on this thesis, has been performed in cooperation with Martin Pichlmair from the IT University Copenhagen, and a work-in-progress paper is to be published in the proceedings of the ACM SIGCHI Annual Symposium on Computer-Human Interaction in Play (CHI PLAY) [Wallner et al., 2015].

Contents

CHAPTER 1

Introduction

Video games as a medium have been on the rise over the past 30 years, and became a multibillion dollar mainstream entertainment industry. With the advent of smartphones in the past decade, digital games have become a ubiquitous part of our everyday culture. Even though video games are already a very rich and diverse medium, they are still in their infancy compared to established media, like for example literature, movies or even TV. *Game studies*, the scientific research of games, is also a fairly young field, and the subfields of *games user research* and *game analytics* have just emerged in recent years.

Interaction lies at the core of this medium, and this work represents basic and exploratory research into understanding the interaction between the player and the game. The goal is to find a formal model for interaction in games that makes interaction quantifiable and that can be used in game testing and game analysis.

During playtesting, for example—a game testing methodology in which a designer directly observes a tester playing the game—the designers have to focus on many things at the same time, like the player's actions in the game, the player's facial and bodily expression, comments and remarks, etc. Having a quantification of interaction can be used as an additional tool, to automatically detect changes in the interaction patterns or unexpected behaviour and to make sure that no issues are overlooked during a test. These issues can, for example, include the player getting stuck or frustrated in the game or when the player presses buttons randomly (button mashing) because the controls are too complex.

The main goal of this research is to make interaction in games measurable and quantifiable. Objectifying interaction in games opens up new analysis possibilities where games, gameplay features or game segments can be analysed and compared based on their level of interactivity. Additionally it can be used to find patterns in games or individual player's actions. A sub-goal of quantifying interaction is to create the necessary and appropriate formal models and modelling approaches that cover interaction in games

well and are expressive enough for practical use.

This thesis is embedded into a wider and interdisciplinary scientific context. It draws from various fields, including computer science, visualisation, human-computer interaction, psychology, information theory and statistics, machine learning, player modelling, game design and games user research.

1.1 Methodology

This work looks at interaction in games by analysing game-controller input through stochastic modelling and by using information theory as the tool to measure the mismatch between the model and the actual user input. This mismatch expresses the amount of abstract information or *surprisal* a certain user input carries. User input that is unexpected and surprising carries more information than input that is fairly predictable and less surprising.

Interaction in games is thus quantified by modelling its underlying low-level processes. Relying on game-agnostic game controller input makes this approach general enough to be applicable to the very large class of controller-based games. All games that use game controllers as their primary input source can be analysed with the tools described in this work. Even though the approach is very general, dynamic on-the-fly training of the models with live data allows the models to be adapted to individual play sessions in real time.

A full set of tools has been created to model and measure the low-level interaction in games. A web-based platform allows models to be trained and evaluated on-the-fly, and the results are visualised in real time. Additional visualisations are also provided to further analyse the models.

These tools have also been used in an exploratory prestudy with a number of different games from various genres. Preliminary results indicate the emergence of certain patterns, that match theories from other and related fields like *action breakdowns* and *routinization*.

1.2 Contributions

This master's thesis represents the first steps into the new field of formally modelling user interaction in games. Even though game play has been modelled before, and information theory plays an important role in many scientific fields, the combination of the two is still something new.

This work contributes to the understanding of interaction in games, modelling of user input and quantifying the model output using information theory. The described approach has been implemented in software and preliminary results from a prestudy are available.

- Information theory is used to measure the mismatch between a formal model and

actual user input. Interpreting this mismatch as information directly corresponds to one goal of game-testing practices: To find unexpected player behaviour that does not match the designers' intent. No existing or similar approaches have been documented in literature yet.

- Formal models of interaction in games are created using discrete-time, discrete-space Markov chains. The models are dynamically trained and adapted in real time and on-the-fly using live data. The problem of estimating the dynamic model parameters from a very small data set is mitigated by considering the full probability distribution of each parameter instead, using a beta distribution.

- The combination of using game-agnostic game-controller data as the model input and dynamically training the models on-the-fly with live data from individual play sessions yields an approach that is general enough to be applied to a very large class of games, but still gives individual results.

- Markov chains are also used as a specific formal model for routinization in games. Routinization describes a process in which actions are repeated until they can be performed almost unconsciously and with little effort. Information content is used as an inverse metric for routinised play. This research was done in cooperation with Martin Pichlmair from the IT University Copenhagen, and the works-in-progress results are to be published in the proceedings of the 2015 ACM SIGCHI Annual Symposium on Computer-Human Interaction in Play (CHI PLAY) [Wallner et al., 2015].

- A series of interviews was conducted to explore the different views on the *playtesting* methodology to act as a guiding use case for the implementation of the tools.

- *Space Walk*, a web-based, real-time analysis and telemetry platform was created as the basis for this research. The platform can be easily extended via plugins and is publicly available online[1]. The platform and all created plugins are open source under the permissive MIT license. Even though colloquial evidence suggests that most larger companies have a proprietary suite of tools, no comparable tools are publicly available or are in a state where they can be easily used out-of-the-box.

1.3 Personal Motivation

My personal motivation comes from my background not only as a researcher, but also as a game designer and developer. Interactivity is at the core of games, but with games being such a diverse medium, it is becoming increasingly hard to define the boundaries between *real* games, interactive experiences, and non-interactive experiences that are still very engaging and immersive.

[1] `http://spacewalk.simonwallner.at/`

In games like the popular first-person shooter-game series Call of Duty,[2] for example, which heavily build on providing a 'cinematic' experience, where is the difference between actually playing the game and watching someone play, especially in segments that are heavily scripted and do not offer much interactivity? How much control and agency (the capacity to act in the game) do we as players still have in these situations, or are we merely watching the avatar on screen?

Another dichotomy in this regard are simple gambling games. In the most basic case, the players don't have any input to the system and have no influence on the outcome. The only choice they have is to play or not to play, yet they are still widely regarded as *proper games*, and some players engage in them very deeply and also feel a sense of agency, the feeling that their actions have an impact in the game.

The implication of these examples is the question of where interaction actually happens. If the players have no input into the game system, how can we still speak of these more extreme examples as games? What happens if we deliberately remove the player's input from the system without telling the players? Will the game still *work*?

These questions arose from a case we encountered during the development of a game prototype that used a simple consumer-grade brain-computer interface (NeuroSky MindWave[3]) measuring concentration levels. By actively concentrating, players could control the game, and even though the system worked very well for some players, it did not work for others, due to technical problems. As a result we had to add the option to spoof the user input for testing purposes. To our surprise, however, the play experience was largely unaffected by this drastic measure. Players still felt that they had a significant influence on the game and felt agency.

In summary, my personal interest in this topic is guided by the following set of guiding questions that go far beyond the scope of this thesis.

What characterises the player-game interaction? How can we describe the player-game interaction and what qualities does it have? What are the differences in interaction between different genres, games or players, what are their commonalities? Where are the limits of interaction, how can we describe them and what implications do they create?

Where does player-game interaction take place? Is actual user input, i.e., the flow of information from the player to the game system, a necessary requirement for interaction, or can a game be perceived as interactive and the player feel agency even if the player has no real input into the system? If we consider such games to be interactive, where does interactivity happen, if not between the player and the game as a system?

How can we make it measurable? Can we make interaction measurable and if so, how can we make it measurable? What conclusions can we derive from the measurements,

[2] `https://en.wikipedia.org/wiki/Call_of_Duty`
[3] `http://store.neurosky.com/products/mindwave-1`

and can it be used to compare games of different genres, or does it even induce a new taxonomy on games?

On the practical side, I want to use my master's thesis and project to contribute to the open-source development of game-development tools. Especially, tools for testing games in early stages of development, where iteration cycles are short and the concepts change almost on a daily bass. Colloquial evidence suggests that most companies make heavy use of such tools during development, but almost none are publicly available, and especially ready to use out-of-the-box.

Over the past decade, we saw the emergence of *indie games*, a segment of this medium that can best be compared to the independent film or music segments. The development of this new branch was largely enabled by both a democratisation of the markets through the introduction of digital distribution and later, by a democratisation of the means of production, most prominently lead by the Unity3D game engine[4]. Indie games play an important role in the avant garde of the medium. More than others, they continue to push the envelope of what games are and what they can be in the future. The tools in this work were specifically created so that they can also be used by those smaller development teams.

I hope that my work contributes to the development of more open-source tools and promotes a healthy sharing culture in regards to those tools.

1.4 Wider Relevance

The relevance of this work is not only limited to games and the analysis of play. The modelling and measuring approach as well as the developed tools can be directly used in related fields.

Advances in predictive modelling of user input can be used in real-time computer graphics for speculative culling and rendering. Predicting where a user looks next, or where they move next in an immersive 3D environment, can be used in caching approaches as a heuristic about what geometry to prefetch and what objects to release from the cache.

In the field of remote rendering, predictive player modelling and speculative rendering is already used to mitigate the round-trip latency in a video-game streaming setup [Lee et al., 2014]. The player's movements and actions are predicted using Markov chains, and the next frame is rendered on the server and sent to the client even before the user input is available. The client then applies corrections based on the actual user input to the frame. In the best case, this drastically reduces the effective latency.

Better understanding of low-level user interaction can also be used as a dynamic aid for 2D pointing devices like desktop mice. Knowing where the user is likely to click next, and having models of the mouse movements on a micro level, could be used to slightly

[4] `http://unity3d.com/`

alter the kinematic trajectory of the cursor's movement, so that it lands closer to its predicted target, thus effectively increasing its target width (cf. [Fitts, 1954; Bootsma et al., 2004]).

1.5 Structure of This Thesis

This thesis is structured in 2 major parts. The first part, *Space Walk* (Chapter 2) describes the technical details of the web-based platform that was created as the testbed for this work. This part describes the underlying use case *playtesting* that guided the requirements elicitation process of this tool, and goes into detail about the plugin infrastructure as well as the various developed plugins.

The second major part of this work is *Modelling Play* (Chapter 3). In this part the theoretical basis of formally modelling play is discussed and developed, related work in that area is discussed, and the theoretical details of the implementation in the *Information Flow* plugin are discussed.

In Chapter 4, preliminary results of the prestudy with a wide range of games are presented and discussed in the subsequent chapters, where this thesis is also brought to a conclusion.

CHAPTER 2

Space Walk

This chapter describes the technical implementation that serves as the base infrastructure for this work. A web-based and real-time platform, called *Space Walk*, has been created to implement and visualise the formal model developed in this work (see Figure 2.1). The goals of this work and the fact that no similar or suitable tools were available made it necessary to create this tool. Additionally, the tool had to suit the research approach and support exploratory workflows.

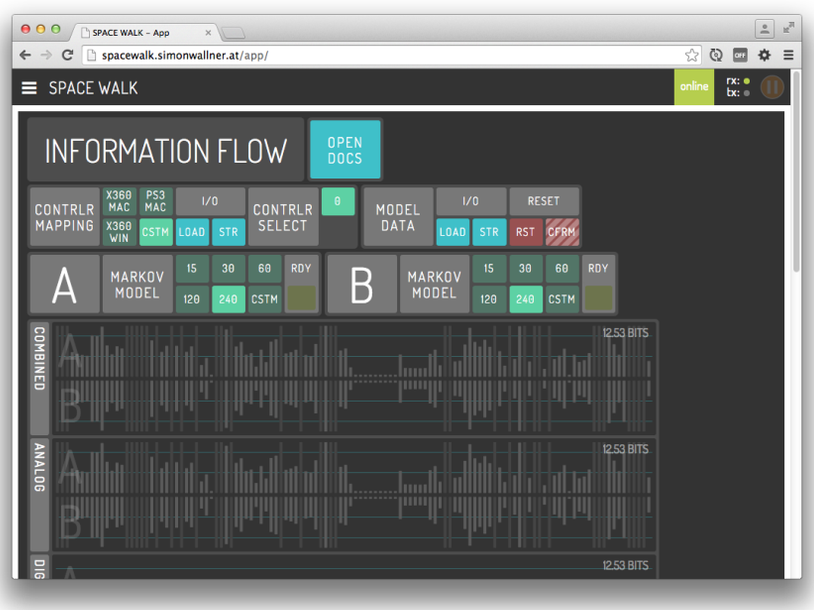


Figure 2.1: A screenshot of *Space Walk* with the *Information Flow* plugin loaded. The plugin features rich visualisations that update in real-time.

The development of Space Walk has been guided by the use case of *playtesting*, a testing and analysis practice where developers and designers sit down with a tester to play certain segments of a game. This use case was chosen as a guidance to elicit the right set of requirements for the platform and to make sure that the tool had a practical relevance for game development beyond the scope of this work.

The following sections give an overview over the platform's structure, review existing systems and also detail the playtesting use case and the resulting set of requirements. Details about the technical implementation are given and an overview over the implemented plugins is presented.

2.1 Motivation

A strong personal motivation in this project was the urge to contribute to freely and openly available game development and analysis tools. Programming, not only in games, is almost unimaginable without using open source tools or programming libraries. There is no reason why this culture of sharing should not also extend to game analysis tools. While larger companies have the resources to buy into commercial solutions or develop their own tools, smaller developers, indies, students or hobbyists usually do not have that kind of budget. As mentioned in the introduction, it is the fringes where the medium grows and thus good tools should be available for everyone.

In this spirit, I hope that my work inspires others to also share their tools. Space Walk explicitly builds on sharing and has the needed sharing infrastructure built right into its core. This could be the seed that will one day blossom into a full toolset of connected open-source tools for game development and analysis.

2.1.1 Existing Systems

Even though colloquial evidence indicates that most larger game companies use analysis tools during the development of their games, only a few are documented in literature and even less are publicly available. From those that are left, none was found that was still actively maintained and also none was suitable for this research work.

Some existing systems are documented in literature, with Microsoft's Game Research's *TRUE* system [Kim et al., 2008] probably being one of the largest. It was created to the needs of running larger scale user tests and gathers all kinds of data for analysis. The system records *User Initiated Events* as a data stream with timestamps attached to each event. Additionally, video is recorded that can then later be cross referenced with the data, as an additional data source during analysis of single cases.

Data Cracker [Medler et al., 2011] is a system that was used on *Dead Space 2*, a very large commercial video game. It provides visualisations of large bodies of event-based data. McCallum and Mackie [2013] present an API that goes in a similar direction, but is aimed at small to midsize projects. Log information is sent to a server that then also provides a visualisation and analysis via a web interface.

In a different direction, *PLATO* [Wallner and Kriglstein, 2013] uses a graph-based visualisation and analysis of time-discrete and time-continuous games.

Besides these documented examples given here, the area of *game analytics* or *game metrics* has been on the rise in recent years. With the rise of the free-to-play business model over the past years, these games have been seen more as services with different requirements on analytics software. Many commercial analytics and metrics solutions are offered online as a service and focus on statical analysis of larger volumes of player data. Notable mentions in this area are GameAnalytics[1], GameSparks[2] as well as the generic Google analytics platform[3], which is also widely used. Important books in this area are the recent "Game Analytics" [Seif El-Nasr et al., 2013a] and "Game Usability" [Isbister and Schaffer, 2008].

Even though analytics and metrics are a seemingly hot topic in game development, no references to truly real-time systems could be found that offer data collection and analysis on-the-fly and are publicly available or even open-source.

Thus, the first logical step when starting this project was to use general-purpose software packages that are not directly made for game analysis. In the early stages of this thesis, the use of R[4], Tableau[5] and especially Matlab[6] has been evaluated. While R and Matlab are general and expressive enough to do any kind of analysis, the workflow of static data analysis did not fit the exploratory research approach. Both also lack the swiftness and flexibility of the visualisation possibilities using modern web-technology. Using web-technology also allowed us to iterate much quicker and to create richer and more appealing user interfaces at the same time.

A full overview over related work that goes beyond existing technical systems and also covers the theoretical foundations of game analysis, games user research and modelling is presented in the following chapter in Section 3.1.

2.2 Playtesting

In order to act as guidance and to make sure that the results of this work are actually relevant and applicable for real-world game development, an exemplary use case was chosen to guide the development. It helped to identify the problems and challenges of the development process and provided a lot of information on how to approach these in the development process of the tool.

The use case selected is *playtesting*, a testing and analysis practice that can be used from very early until late in the production cycle of a game. The standard development

[1] http://www.gameanalytics.com/
[2] http://www.gamesparks.com
[3] http://google.com/analytics/
[4] http://www.r-project.org
[5] http://www.tableau.com
[6] http://mathworks.com/products/matlab/

model in game development follows an iterative approach. New game ideas are incubated by creating a number of prototypes that explore the many different aspects of the idea. Prototypes can have many different forms, but as a rule of thumb they become more elaborate the further a project progresses. In very early stages, even simple techniques like *paper prototyping* can be used, where game ideas or elements thereof are tried out quickly by recreating them through the use of cards, tokens, crafting material and so on. In later stages, prototypes are often created with the production tools, to try out new mechanics or to see how changes impact the gameplay.

Game Development can be seen as a constant exploration of the game space. The game is changed and improved constantly up until the final release. Appropriate tools are needed to support this highly dynamic process.

2.2.1 Defining Playtesting

There is no canonical definition of the term *playtesting*. It is a technical term that emerged from the practice, and the definition of what a playtest is varies widely from very narrow and distinct to rather general.

Schell [2009] describes playtesting as:

> [..] playtesting is all about getting people to come play your game to see if it engenders the experience for which it was designed. [Schell, 2009, p. 390]

He further goes on describing the exploratory nature of playtesting:

> Anyone can find things they know they are looking for – but only a truly observant designer, who has learned to listen deeply to players, can find the things they don't know they are looking for. The key is to keep your eyes open for *surprises* [emphasis in the original]. To be surprised at a playtest, you must already have ideas about what will happen: players will attack level two a certain way, they will get excited at the start of level three, etc. Whenever anything out of the ordinary happens, good or bad, be ready to jump on it, and find a way to understand it. [Schell, 2009, p. 396]

Amaya et al. [2008] characterise playtesting as follows:

> In Playtesting, we focus on players' opinions to illuminate areas of the game in which player experience does not map onto design intent. By explicitly tapping players' opinions and attitudes, we hope to reveal more about what the subjective experience of playing the game is like for people and ensure it matches with what the designers hoped it to be. [Amaya et al., 2008, p. 41]

They go on describing how playtesting changed for them over the years.

> The new goal of Playtest focused on how to make a game better as opposed to evaluating how good or bad a game was. Hence, early Playtesting and early iteration in a game's production cycle were stressed, and Playtests were focused on more experimental and exploratory questions [...]. [Amaya et al., 2008, pp. 42]

The biggest difference between these two views on playtesting is that Schell regards the process mostly as qualitative whereas Amaya et al. use playtesting to obtain quantifiable insights from a larger group of testers.

To further explore playtesting, a number of interviews with game designers and developers from Austria, Denmark, France, the UK and the US were conducted to define the term and to find out how it us used in practice. Transcripts of selected interview segments can be found in Appendix A. These interviews also illustrate the individual development practices, and they are especially recommended to readers who are not already familiar with game development.

2.2.2 Working Definition

Based on the literature, the interviews and my own experience as a game designer, I define playtesting as follows:

> Playtesting is a testing and analysis practice in which the designer(s) directly observe the tester(s) playing a specific section of a game or game prototype. The goal of a playtest is to find unexpected player behaviour that does not match the design intent, or that is the symptom of problems in the game. Additionally, playtests are used to see what emotions the game invokes in the players.
>
> Playtesting is an exploratory practice and it can happen ad hoc and does not require a strict protocol. It is interactive in the sense that the designers and players can ask questions if needed, and the designers can react to the tester's actions immediately by changing the test setup or providing help or hints. The implicit analysis takes place during the test or immediately afterwards.

Additional protocols like the *think aloud* protocol can be employed, where the testers are asked to verbalise all their thoughts and motives during the test (cf. [Hoonhout, 2008]).

2.2.3 Challenges in Playtesting

Playtesting is an ideal guiding use case for the development of Space Walk, because it highlights a few challenges and problems in the development and analysis process. Playtesting can be challenging since it requires observing the players and their actions at

the same time as monitoring their progress in the game. How are they performing in the game, what buttons are they pressing, what are their verbal comments, what are their bodily reactions to the game, are they bored or excited, etc.? This load of concurrent information leads to the situation where a designer has to decide where they want to focus their attention or to constantly switch between the various aspects.

Another challenge is that not all information can be directly observed. The most basic example is that a designer cannot observe a player's input simply because the view to the game controller is obstructed. But even if this is not the case, recurring or complex patterns can be hard to observe for humans, but pose only a small challenge for a machine if the patterns can be formally described.

Playtests focus on individuals and their individual reactions. It is a time-consuming process that requires the designers as the core resource. This also means that playtests are most often only performed with a small number of testers. The insights gathered are thus individual for each test session and cannot be directly generalised to larger populations.

The interviews and earlier conversations with developers also revealed that in many cases developers do not have the time for an in-depth analysis of recorded test sessions and often skip it completely. Analysis happens during the test session, or immediately thereafter, sometimes aided by post-test interviews or discussions with the testers.

In a similar notion, there is often not the time to set up more elaborate testing procedures. Especially in early stages, the game changes so rapidly that a study would already be outdated by the time it was drafted. Special builds are usually created for a playtest, but playtests can also happen more spontaneously and ad hoc.

2.3 Requirements

Based on the description of playtesting and its challenges, I propose a system for exploratory analysis that is easy to use, requires little setup, is easily extensible and modifiable and available online as open-source software. The system shall fulfil the following requirements:

Exploratory post hoc analysis The system shall support exploratory and post hoc analysis. This stands in stark contrast to other hypothesis-driven approaches, but in many cases, the game that is to be analysed changes too quickly during development to craft hypotheses and as Schell puts it, the challenge lies in finding the things we weren't looking for in the first place [Schell, 2009, p. 396].

Focus on individual players A connected requirement is to focus on individual players, instead of analysing larger test groups or general populations. One way to still arrive at meaningful results is to generate more data from single testers instead of only sampling some data from a larger group.

Ephemerality Not having the time or the resources to further analyse the test data after the playtest has concluded was a recurring topic in the interviews. As a consequence, an interactive and real-time system is proposed that provides the results on-the-fly. However, this does not mean that the data cannot be recorded for later analysis, but rather that this becomes an optional step.

User friendliness The system shall be easy to use and have a simple yet rich user interface. The overhead of using it should be minimal so that it can be used quickly without a longer setup time.

General and special-purpose elements Some aspects of this system will remain constant for all games, and some need to be tweaked, modified and adjusted to fit a specific game or use case. By using a plugin system, the common software elements can be integrated in the core, while the system can be easily extended with custom-made plugins.

Software-as-a-service To reduce the overhead of maintaining the software and to simplify the deployment process within organisations or in the development community, a software-as-a-service model shall be adopted both for the core system and the plugins. It shall be easily and universally accessible via the web to further reduce setup costs. Providing the software as a service online also makes it cross-platform by default.

Sharing infrastructure In order to support the open sharing of game-development tools, the system shall include mechanisms that facilitate sharing of plugins as a first step and leading example in that direction.

Common communication protocol In a similar notion, the system shall use an open communication protocol between its various components. The protocol shall be human readable and have a standardisation and extension mechanism built-in for continuous extension and improvement.

2.4 The Space Walk Platform

The discussed requirements are implemented in *Space Walk*, a web-based, real-time telemetry and analysis platform developed under the permissive MIT open-source license. The system is available online[7].

The Space Walk platform is built on three main pillars: A **common communication protocol**, the **web-based application** and a **plugin system** that allows developers and researchers to quickly create new plugins or adapt existing plugins to their needs.

[7] `http://spacewalk.simonwallner.at/`

2.4.1 The Protocol

Communication between a game and the Space Walk web application is realised over the network and uses a custom application-level protocol. Communicating over the network allows Space Walk to be used with games on any host platform, be it a standard PC, a mobile or embedded device or a game console. As long as the game has basic network access, data can be sent and received.

An in-depth description of the protocol and its features is available online[8].

WebSockets[9] are used as the underlying network protocol. WebSockets are a recent standard that especially allows web browsers to use persistent network connections. WebSockets are the only standard way for a browser to create such a connection. They are implemented in all major browsers and can be used across many different platforms including mobile.

Space Walk uses an application level protocol that is based on exchanging new-line terminated JSON messages. JSON, the *JavaScript object notation*, has gained popularity in the last years and became the de-facto standard for communication in web-based applications.

JSON was chosen for its simplicity and readability. If designed properly, messages are self-documenting and easy to understand. Using a verbose text-based protocol of course has a performance penalty in both size and computational cost, but the need to be able to create and modify plugins quickly and often definitely outweighs these costs. See Listing 2.1 for an example message.

```
1 {
2     "type": "core.simpleLog.message",
3     "payload": {
4         "level": "info",
5         "message": "hello Telemetry!"
6     }
7 }
```

Listing 2.1: Example message (newlines added for readability)

The basic message structure is quite simple. It has a unique `type` id and a `payload` field that encapsulates the arbitrary payload. The protocol should be self-describing and easy to understand. The different message types form protocol features. Each feature can be implemented independently of others. A remote logging application for example, can only implement the needed features and ignore all other messages.

Space Walk aims to create and use a common and standardised protocol that is used between all the plugins, but that is also useful between other applications outside of the

[8] `http://spacewalk.simonwallner.at/protocol.htm`
[9] `https://en.wikipedia.org/wiki/WebSocket`

Space Walk context. To achieve this interconnectivity, the protocol is standardised and the standardisation process is built into the protocol.

The protocol supports namespaces for message types to avoid name collisions and to provide a mechanism for continually extending and improving the protocol, while allowing developers to extend it for their own purposes.

Space Walk comes with two reserved namespaces: `core` and `ext`. The `core` name space is meant to be stable throughout a major version of the protocol. New features are first introduced as an extension in the `ext` namespace and then move into the core once they have stabilised enough and proven useful.

Outside of the reserved namespaces, developers can use custom namespaces to avoid naming conflicts and to better structure features. A reverse domain name notation scheme is recommended for developers who are extending or creating their own features and message types. See Listing 2.2 for an example.

```
{
    "type": "at.simonwallner.biometrics.data",
    "payload": {
        "HR": 123
    }
}
```

Listing 2.2: custom name space example

All messages must be valid JSON objects and all parameter names should be in camelCase starting with a lower case letter. The reverse domain part of the name space should be in lower case followed by further sub name spaces, feature names and message names in camelCase, e.g., `at.simonwallner.spaceWalk.myFeature.messageName`. All parameter names and message types are case sensitive.

2.4.2 Space Walk Application

The web-based Space Walk application forms the second main pillar of Space Walk. It is the front-facing core of the project, which provides the plugin infrastructure and maintains the network connection.

The browser was chosen as the technical platform for its ubiquity and the ease of deployment of web-based services. Web browsers pretty much run on every device, and through modern web services, updates to the platform as well as to the plugins can be deployed to all users in just seconds with a single click. This takes away the need of maintaining individual installs and making sure that everyone is on the latest version.

Space Walk is created with JavaScript, the much loved and hated programming language of the web (cf. [Crockford, 2008]). Compared to other compiled and interpreted languages, JavaScript and the browser is definitely not the fastest option, but it allows

for very quick iterations. It is beginner friendly, and with JavaScript and HTML5 it is very easy to create rich user interfaces in the browser.

Data visualisation in this project is done with D3.js [Bostock et al., 2011]. D3 is a very versatile library that makes working with data easy using its data-driven approach. Modern web browsers are fast enough to render even large visualisations with large data sets at interactive speeds.

Space Walk communicates over the network and in the most basic case, the game sends its data to the web app, which manages the network connection, and then forwards the data to all loaded plugins in the browser. This data flow also works the other way round so that plugins can send data to the game, for example, to tweak in-game parameters, request screenshots, perform certain actions etc. See Figure 2.2 for the communication diagram.

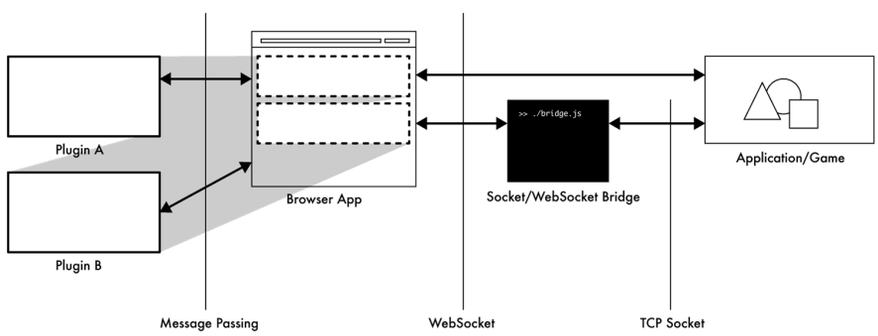


Figure 2.2: The communication flow in Space Walk. The game acts as a server to which the Space Walk application either connects directly or through a Socket/WebSocket bridge. Received data is then relayed to all loaded plugins within the application. Communication is supported in both ways.

The game creates a server (web)socket on the default port `60600` and waits for connections. By acting as a server, different services can connect to the game concurrently. A developer can connect to the game with a simple remote logger plugin, while someone else is listening to the data with an analysis application, while an additional small tool might also connect and just record all messages to disc for later analysis. All data is broadcasted to all connected clients.

Since WebSockets is a fairly new standard, built-in support in programming languages and standard libraries varies a lot. To make development easier and to be able to use simple TCP sockets instead, a small tool was created with node.js that acts as bridge between a WebSocket and TCP socket application. The game then just creates a TCP socket server on the default port `60601`, the bridge connects to it and then offers the service as a WebSocket server. The bridge can be downloaded from the website[10].

Once the game is up, the web app can establish a connection to the game and relay all data as-is to the loaded plugins via the JavaScript `window.postMessage` API.

[10] `http://spacewalk.simonwallner.at/tools.htm`

2.4.3 Plugins

The plugin system forms the third and last main pillar of Space Walk, and plugins are the main development and deployment model in Space Walk. Plugins have been chosen to separate the general purpose elements of the web application from the game and use-case-specific special-purpose elements that are then used to analyse the individual games.

Plugins also facilitate the creation of new features, or the modification and extension of existing features to suit a certain project. Plugins can be deployed and shared easily, an aspect that is explicitly supported by being able to load plugins from anywhere on the web.

Plugins themselves are just simple web pages that are then loaded into the Space Walk web application as `iframes`. The communication with the app is done via the `lib-space-walk.js` library that has to be included in the plugin. This library provides an `onMessage` callback to receive data and a `postMessage` function to send messages back to the game. Cross communication between plugins is not possible.

Since plugins are nothing more than static web pages, they can simply be hosted and deployed anywhere on the web. For instance, through github pages, Amazon s3 or any other static http hosting service. To load a plugin, users only have to enter the plugin's url or select a plugin from the built-in list of recommended plugins.

Since the files are web hosted, updates to Space Walk or any plugin are directly visible to the users, and they always work on the latest version. There is no need to manually manage updates or installed versions.

2.4.4 Prerequisites

Even though Space Walk is easy to use, there are some prerequisites that need to be fulfilled. First and foremost, the analysed game must be able to create a socket/WebSocket server and implement the protocol features that are required. In most cases, the implementation of the server component can be very simple and will only be a few hundred lines of code in most cases.

If it is not possible to add Space Walk support to the game directly, additional tools can sometimes be used to capture the required data. In the case of the Information Flow plugin, described in detail in the next chapter, only data from the game controller is needed. A small tool was created to independently capture that data, so that the plugin could be used with any commercial game without modification.

2.5 Implemented Plugins

A number of different plugins have been implemented over the course of this thesis. The first plugins have been created to debug the system and as a first proof of concept, leading up to the Information Flow plugin that will be covered in depth in the following chapter.

The Information Flow plugin implements the theoretical core contributions of this thesis, and it was also used in the prestudy described in Chapter 4.

Message Debugger The message debugger is just a simple debug view that prints all received messages to the browser window. It is useful in debugging new features or new game integrations, and the codebase can also be used as a minimal starting point for new plugins. It basically represents the *hello, world!* of Space Walk.

Raw Input Debugger The raw input debugger was created to debug data received form game controllers and directly visualises all input data. Unfortunately, there is no standardised game controller layout, and USB HID drivers just offer a collection of analogue and digital inputs. The indices of these inputs are different for most combinations of controllers/operating system/input libraries. This plugin was thus needed to find the mapping for each controller.

Direct Controller Vis Based on the mapping obtained from using the raw input debugger plugin, a direct controller visualisation was created that simply shows the state of all buttons, analogue sticks and triggers on the controller (Figure 2.3). This plugin is especially useful during playtests to see what buttons the players are pressing in real time. This plugin was also used in the exploratory study on interactivity and routinisation in games (see chapter 4).

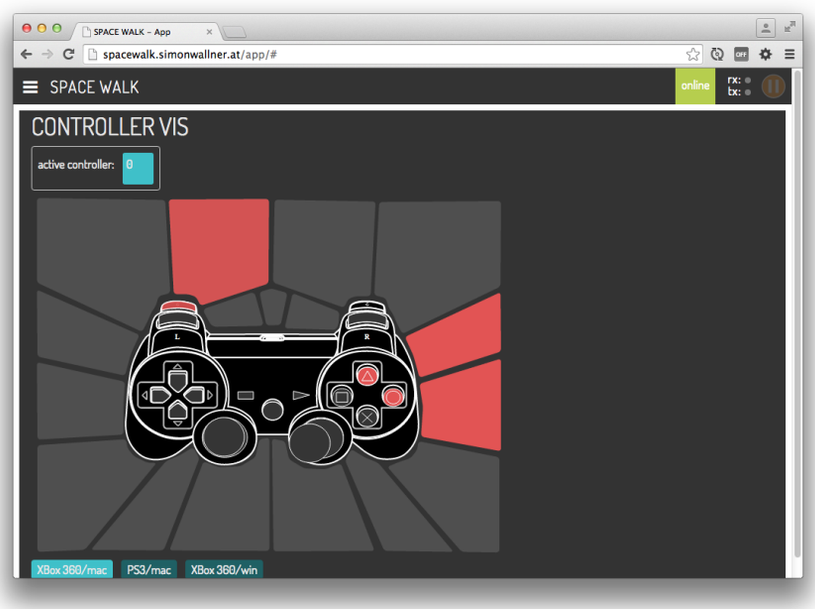


Figure 2.3: A direct visualisation of the controller input in real time. This plugin can be used as a handy helper in playtests to exactly see what buttons players are pushing and how they interact with the game.

Input Complexity The input complexity plugin is a generalisation of Swain's static analysis of input complexity [Swain, 2008] (Figure 2.4). Each active axis or pushed button is counted to produce a scalar complexity value. The original paper does this statically for a game or game segment, but this plugin generalises this approach to real-time applications by counting the number of different inputs over the last n seconds. Different filters are then applied to the data (box, saw, Gaussian), and the data is visualised as a simple rolling bar chart.

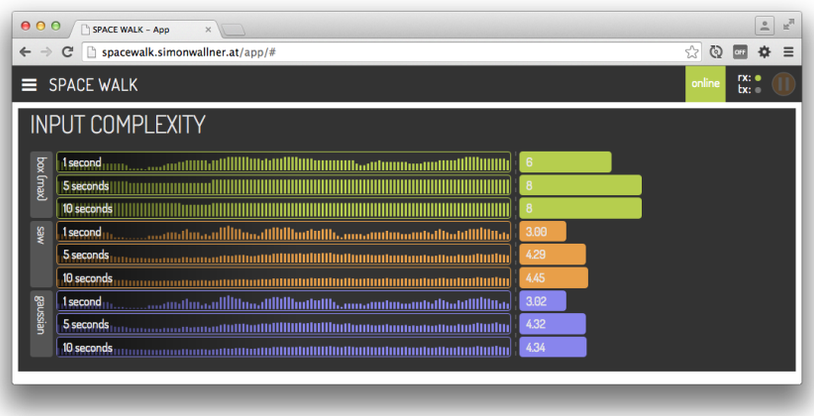


Figure 2.4: The Input Complexity plugin. The scalar complexity value is compute in windows of variable length, and different filters are applied to the data.

Information Flow The information flow plugin quantifies interaction in games by modelling player input in real time using Markov chains. The results are also visualised in real time, and the plugin has additional features to visually explore the models. (Figure 2.5). The plugin allows using models with different parameters and also comparing the results from different models. The following chapter goes into details about the theoretical foundations of the formal model, implementation details and a general description of the plugin.

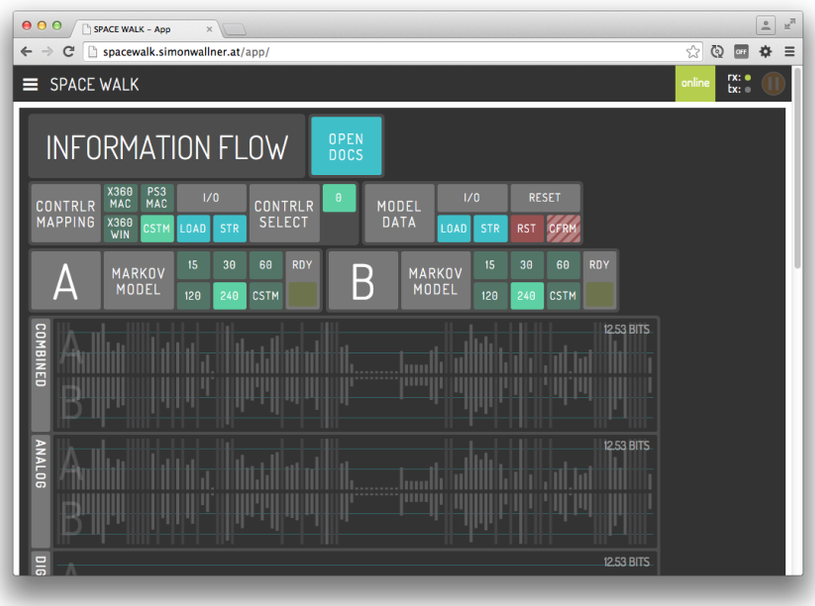


Figure 2.5: The Information Flow plugin quantifies interaction in games by modelling it through user input.

CHAPTER 3

Modelling Play

This chapter discusses the theoretical basis for modelling play as well as the challenges that arise during the implementation of these models. Modelling play means that a formal description of play is found that can then be used as a surrogate in analysis and testing. Through formal modelling, play is also objectified and quantified. It allows us to measure play and to compare the results between players, games or genres.

By modelling play, however, we cannot completely capture the underlying domain with all its aspects. Models are abstractions and thus, by definition, always have an intrinsic modelling error.

Through the following examples I want to motivate formal modelling of play as a practical analysis tool for games. The examples presented here are to illustrate the wide range of existing modelling approaches. The first example quantifies naturalness of user interfaces by annotating user test data using qualitative criteria. On the other end of the spectrum and not even a model by definition, an isomorphic transformation of game rules reveals familiar structures.

Example: Modelling Naturalness The first example from the field of Human-Computer Interaction (HCI) investigates the naturalness of an experimental user interface by modelling the naturalness through *action breakdowns* [Gamberini et al., 2011]. An action breakdown is described as an interruption or slowdown of an interaction caused by a crisis that is then immediately resolved by a reevaluation of the situation. In this study, the researchers analysed locomotion devices for navigation in virtual environments and action breakdowns occurred when the users stopped in their path to reorient themselves. The qualitative description of the action breakdowns was used in annotating video recordings of the study to quantify the number of action breakdowns for each test trial. This model was then used to compare the results from different user interfaces and to further analyse the test results.

Even though the description of an action breakdown is only based on low-level features of human locomotion and behaviour, it is successfully used to quantify the high-level concept of *naturalness* of user interfaces. △

Example: Magical Tic-tac-toe The second example is on the opposite end of the modelling spectrum. In contrast to the qualitative criteria that was the foundation of the previous example, this example uses a formal transformation to reveal familiar structures in a game. The transformation of the game rules is even isomorphic, and thus by definition it is not a model.

The game whose rules we will transform is a simple math game in which players take turns in picking a number from 1 to 9. Each number can only be picked once, and the player who can first form a sum of 15 with three of the picked numbers wins. See Table 3.1 for an example.

numbers	~~1~~, ~~2~~, ~~3~~, 4, ~~5~~, ~~6~~, ~~7~~, ~~8~~, ~~9~~
Alice	2, 6, 3, 8
Bob	7, **5**, **1**, **9**

Table 3.1: Example game of *Magical Tic-Tac-Toe*. Alice and Bob are playing. Alice has the first pick and Bob wins after the fourth turn.

The game might seem quite confusing at first, keeping track of all the numbers and repeatedly computing sums. The name already suggests, however, that there is a similar game to which all aspects of the game can be mapped 1:1. This game is classical Tic-tac-toe, as the following table shows (see Table 3.2). If we take the available numbers from 1 to 9 and arrange them into a magic square, we can clearly see that every action in the original game corresponds to a move in Tic-tac-toe and vice versa.

2	7	6
9	**5**	**1**
4	3	8

×	◯	×
⊖	⊖	⊖
	×	×

Table 3.2: The game transformed in to the Tic-tac-toe form.

Using this transformation, the game is already well visualised, and the players' strategies become immediately apparent. The game also shows that math skills, the ability to sum up single digit numbers, are not at all required to play the game. △

These two examples illustrate the usefulness of formal modelling as a tool in game or play analysis. The approaches of modelling play are plentiful and different models encompass different aspects of games and the interaction with games.

Additionally to using models directly as tools for data analysis, we can also interpret the models themselves as data and perform a meta analysis on them. Seen as data, the

models are influenced by our prior knowledge and assumptions inherent in the research approach and they are also influenced by the concrete data they are based on.

An abstract excursion into history that illustrates the importance of the models themselves is the heliocentrism/geocentrism debate from the 16th century. If we look at the models themselves, what can we learn about the scientific body of knowledge at that time, the predominant belief, religious and philosophical views? Closing the loop to games, or more generally interactive media: The modelling approach used can reveal a lot about the base assumptions on and the structure of gameplay.

In this work, play is formalised and quantified by modelling user interaction in games at the input-level. Play is the activity of performing or *playing* a game. Games as an interactive medium build on interaction, and player input is one crucial element of interaction. Through modelling low-level user input, we can model the higher-level concepts of interaction and play in general.

3.1 Related Work

The related work in modelling aspects of video-game play is spread out over many different disciplines and sub-fields. Relevant work can be found in computer science, human-computer interaction, artificial intelligence, psychology, design studies, data mining and analytics or more practical industry-driven case studies.

Game studies, the scientific research of (digital) games, has been sharply on the rise in the last decade with the main conferences founded starting in the early to mid 2000s[1]. Applied game studies is similarly a new field with seminal books also released in the recent years (e.g., [Juul, 2005] and [Salen and Zimmerman, 2003]).

A general and holistic model for game design and game research is the *MDA Framework* [Hunicke et al., 2004], a model that offers approaches to game analysis on different levels. A game is separated in the three layers of **M**echanics, **D**ynamics and **A**esthetics, and the interconnections between the three are explored.

In the recent years, the subfield of *games user research* and *game analytics* also saw a rise in interest in part lead by the the advent of new monetization models like free-to-play, which interpret games as a service that needs to be constantly optimized to increase per-user revenue (see [Isbister and Schaffer, 2008] and [Seif El-Nasr et al., 2013a] as illustrative examples).

Looking at more formal models of players and play, a good and general introduction into the topic is presented by Canossa [2013]. He starts with a general discussion about what constitutes game analysis: "Analyzing game-related data, at its core, is a process that involves being able to articulate knowledge and meaning from apparently meaningless data." [Canossa, 2013, p. 255] and even bring Wittgenstein's *Tractatus Logico-Philosophicus* to the table, interpreting his popular quote: "the limits of my

[1] first DIGRA in 2003, first FDG in 2006 (fka. GDCSE), first CHIPLAY in 2014

language mean the limits of my world"[Wittgenstein, 1922, p. 74] in the sense that "In the specific case of game data analysis, the verbs used to talk about player behaviour are defined by the game variables measured and tracked by the telemetry system. These variables, once measured, become metrics, and from metrics, features are extracted; the selection of which features to use is a pivotal component of game data analysis." [Canossa, 2013, p. 256]

The paper further explores the knowledge-forming process through different stages from variables to features to final models. The question of what variables to track is both looked at from a conceptual and technical point of view with respect to introduced structural bias, exploratory analysis and technical feasibility.

Yannakakis et al. [2013] present a taxonomy of modelling approaches for *player modelling*, a term they very narrowly define as: "the study of computational means for the modeling of player cognitive, behavioral, and affective states which are based on data (or theories) derived from the interaction of a human player with a game" [Yannakakis et al., 2013, p. 46].

This taxonomy partitions the field into *top-down*, *bottom-up*, and the overlapping area of *hybrid* approaches. Top-down modelling starts with a theoretical framework, from which models are derived and then validated against observed data. On the other hand, bottom-up approaches start with the elicitation of data and then form models exploratively after the fact. Hybrid approaches fall in between the two, and guide the model crafting from data by adding additional knowledge, initial assumptions and theoretical frameworks. My work about modelling user input with Markov chains described below falls into this third category.

In a similar direction Smith et al. [2011] present an inclusive taxonomy that spans 4 major categories that go beyond top-down and bottom-up approaches. They performed a larger survey to classify the works and to identify areas that are over/under represented in current research (as of 2011).

Yannakakis et al. [2013] strongly focus on the player's experience and cognitive states and heavily rely on artificial intelligence (AI) and machine learning methods in their work. They mention the modelling of non-player characters (NPCs) as a synthetic testbed for models, since the actual behaviour of the NPC is explicitly available in code, and can thus be used to compare it to the model. Holmgård et al. [2014] take this a step further and model play through AI agents that are then analysed. They function as an abstract surrogate that is used to interpret the human decision making process.

Biometrics and approaches based on objective physiological data are also well documented. Works in this area often have a close relation to psychology, where biometric studies have a longer tradition. A good introduction into the field is given in [Nacke, 2013], explaining numerous physiological parameters that can be tracked, their advantages and disadvantages and a large literature overview. Sundstedt et al. [2013] additionally cover the use of eye tracking to analyse the player's visual attention in 3D virtual environments.

Drachen et al. [2010] present a case study on the correlation of heart rate and galvanic skin response (GSR) with self-reported, subjective player experience in games. In a similar direction Martinez et al. [2013] use advanced machine learning techniques (Deep Learning) to explore the relation between GSR, blood volume pulse and a self-reported pairwise preference. The paper provides a good introduction into machine learning methods in this area, and extensively compares the deep learning results with other methods.

A topic that has gained much interest over the last years, partly driven by the proliferation of the free-to-play and thus software-as-a-service model for games, is analytics and metrics. The boundaries are not all that clear-cut, but a large part is focused on analysing large volumes of data during production or post launch with the goal to improve on the game design, monetisation and player retention.

The recent book [Seif El-Nasr et al., 2013a] covers the field in depth, and [Drachen and Canossa, 2009], in an article from 2009, a time before free-to-play became so ubiquitous, provide a general introduction from an HCI perspective.

Colloquial evidence indicates that almost all larger game development studios use some form of analytics or metrics, but only a handful of them are documented in literature. Probably the largest one is Microsoft's *TRUE* system [Kim et al., 2008] and other notable examples are [Medler et al., 2011] and [Lynn, 2013].

In the area of industry lead case studies, data from the action adventure game *Tomb Raider* is analysed in [Mahlmann et al., 2010] to predict the play duration after which players will stop playing based on data from the first levels. A case study from the racing game *Project Gotham Racing* is covered in [Hullett et al., 2012, 2011; Zimmermann et al., 2012] with a focus on game usage and player progression in the game.

In the area of adaptive gameplay, Yannakakis and Hallam [2009] use artificial neural networks and a gradient ascent scheme to optimise the perceived *fun* of a digitally enhanced physical kids game. After each round (90 seconds) the game parameter is changed based on data recorded during the session.

Another area of work is the spatial analysis of games and players. Drachen and Schubert [2013] and Seif El-Nasr et al. [2013b] give an overview and intro into this subfield.

Not much literature exists in the area of modelling low-level interaction or user input, especially not in games. A classic in computer science and human-computer interaction is Fitts's 1954 work on "The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement" [Fitts, 1954]. In this seminal paper, Fitts models the relation of parameters of a target acquisition task. The core of the work is *Fitts' Law*, a formula that expresses an *index of difficulty* in *bits*, which linearly correlates with the task execution time. Fitts makes heavy use of information theory, but does not give a mathematical justification for his formula.

Extending Fitts' work, Bootsma et al. [2004] look at motion on a micro-level and

analyse the kinematic patterns in goal directed movement. Another work that goes in the direction of micro-analysis of user interaction is [Gamberini et al., 2011] where the naturalness of locomotion devices in virtual environments is analysed through *action breakdowns*. An action breakdown is defined as a slowdown or interruption of the user's actions in order to recover from a cognitive crisis.

In applied game analysis, [Guardini and Maninetti, 2013] analyse the user input in a rally racing game at a sampling frequency of 10 Hz to gain insights about the dynamic complexity and difficulty of different track segments.

Probably the closest related work is the *Outatime* project [Lee et al., 2014], which aims at effectively reducing the round-trip latency in remotely rendered games (i.e., video-game streaming services) by speculative rendering based on input prediction. They use individual per-user off-line trained discrete-time, discrete-space Markov chains to model the user input in the game. Their work also features techniques to mitigate misprediction errors and provides a very extensive evaluation of all relevant parameters. This work is also the strongest indicator that discrete-time, discrete-space Markov chains are a viable model for user input.

The literature presented here serves as an important theoretical basis, but except for [Lee et al., 2014] no references to modelling approaches could be found that are directly related to the approach presented in this thesis. Even on a more general level, no references to work could be found that applies information theory in any form in the domain of game analysis.

3.2 Overview and Methodology

This chapter progressively develops a formal model of play and interaction in games step-by-step from more general concepts to specific implementation details. The final goal of the model is to measure and quantify interaction in games in real time, by measuring the mismatch between a dynamic model of user input and each user action.

The input into the model is game-controller input and since human action is generally assumed to be non-deterministic, the resulting underlying models are stochastic models. These stochastic models take a sequence of game-controller events as their input and output a probability value that indicates how likely the current user input was under the given model.

Information theory is then applied, and these probability values for each game-controller input are used to quantify the model-action mismatch by computing the information content of the input.

Discrete-time, discrete-space Markov chains are proposed as the concrete stochastic model class for this approach. Markov chains assign probability values based on a transition matrix that holds the probability values for each transition from one state to the next. In some applications this transition matrix can be defined top-down based on prior theory, but in our case, the transition matrix is estimated empirically on-the-fly

based on previous game-controller input.

Lastly, this chapter goes into technical implementation details regarding the Markov chains, how to effectively estimate the transition matrix at run time and how to tackle the problems that come from having only very little data available.

The following figure (Figure 3.1) gives a high-level overview of the model. The model takes a vector of game-controller input as its input and outputs a scalar value value of model-action mismatch.

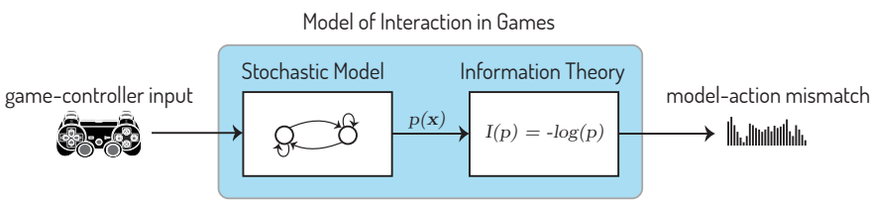


Figure 3.1: A high-level overview of the model. The model takes a vector of game-controller input as input and outputs a scalar model-action mismatch.

Section 4 then presents results that have been obtained with the described model. In this case, the model's output is used to identify interaction and play patterns in a number of games and to find potential issues and problems in their design.

3.3 Modelling Play Through User Input

The domain of the model developed in this thesis is limited to raw user input from game controllers. *Raw* in this context means input that is directly read from the game controller. The input does not have a game-associated meaning like *jump* or *shoot* but is just a collection of digital and analogue values, one for each button or controller axis.

The proposed model is based on the assumption that by capturing the raw user input, 100% of the *information* that is flowing from the user to the (game) system is captured. In this bottom-up approach, the model thus captures all interactions that constitute play.

Basing the model on raw user input alone also makes this approach independent of the actual game. The input does not have to be interpreted in the context of the game (i.e., *jump*, *shoot*, etc...), which makes this approach universally applicable to a wide range of controller-based games.

Game-controller input is the lowest common denominator for a large class of video games. All current-generation video-game consoles use game controllers as their primary input, and most PC games can be controlled with a game controller as well. A large class that is not covered however is the class of touch-based mobile games. Third-party game-controller accessories exist for mobile platforms, but their utilisation is insignificant.

The modelling approach used here uses game-controller input both as its domain (the subject that is modelled) as well as empirical data basis that is used to define the model

parameters. As we will see in following sections, some of these parameters can be chosen up-front, and some parameters are dynamically trained and estimated based on observed data at run time.

The model developed here is of course limited in its scope and does not provide any explanation or interpretation of the resulting output. It always requires additional expert knowledge, from a designer or developer, to interpret the results. The amount of model-action mismatch can only be seen in relation to the model.

Generally, the model-action mismatch can be large for two reasons. First, the game situation changes abruptly and thus the model is still largely based on data from the *old* situation. The new situation elicits different player behaviour that is thus classified as *surprising*. This case is very easy to detect for a designer, because the changes caused by different game segments usually follow the designers intent. Second, the player's behaviour changes abruptly. This is the much more interesting case because the player displays behaviour that goes against the designer's intent momentarily, a strong sign for issues in the gameplay.

A small model-action mismatch on the other hand does not mean, that the gameplay and the player's action completely follow the design intent of the game. Situations can arise in which players show relatively uniform and well predictable behaviour. Even though not captured by a change in the model-action mismatch, these situations can be more easily spotted by designers because they remain stable for a longer time, giving designers more time to analyse. It is the sharp but subtle changes where the model can contribute the most information (also see the description of the results in Section 4.2).

3.3.1 The Syntax and Semantics of Raw Input

By modelling raw user input alone without linking it to the semantics it has in the individual games (i.e., *jump*, *shoot*, etc.), this work specifically focuses on the syntax of user input in games. It is the metaphorical letters, words and sentences of user input we focus on, but is this enough to reason about the high-level concept of interaction in games?

What we can learn through syntactical analysis are the intrinsic structures of user input, and we can model its syntax and the underlying rules or grammar that govern it. We can compare the models of individual players and games to see where they match or differ in various genres, games or play situations. Even though the subjective player experience and the question of weather a game is *good*, *bad* or even *fun* is outside the scope of the model, a game designer or developer can interpret the data, compare the data of different players or different game situations to link the found patterns and structures to elements in the game.

Even though the model is based on syntax alone, by choosing a certain modelling approach we add additional meaning or semantic to the whole system. The approach we use informs the interpretation of the data in a certain way. Thus information or knowledge is added to generate *meaningful* results.

As we will see in the following sections, Markov chains have been chosen as an *appropriate* model for game-controller input. They allow us to model sequential behaviour and are based on a very simple state concept. It is our prior knowledge about games and input in games that lets us argue in favour of Markov chains and by doing that, we add all this information and semantic interpretation and meaning to the model that make the results expressive beyond the pure syntactical analysis.

3.4 Stochastic Modelling

User input can generally be assumed to be non-deterministic. This work thus uses stochastic models to model user input. Stochastic models describe a probability distribution, for example, how likely it is that the player pushes a certain button, a sequence of buttons, or stops to interact at all.

The model takes a sequence of user input as its input, and for each input event outputs the conditional probability of the current event based on the previous inputs $p(x_t|(x_{t-1}, x_{t-2}, ...))$, where x_t is the current event and $(x_{t-1}, x_{t-2}, ...)$ is the sequence of previous inputs. With this model, we can quantify the expectancy of a given input event in a given situation. In other words and seen from a different angle, the model predicts user input with certain probabilities.

In the context of game development, we can find a nice analogy between the stochastic model and the designer's knowledge and experience: Based on knowledge and experience, designers have a certain expectancy of what a player will be doing in a game. They can predict their actions to some extent, and the more time designers spend with a game or a player, the better they understand their actions and maybe even motives. From observations they learn their way of playing and are able to anticipate their actions.

The analogy to that in stochastic models is the information or *knowledge* intrinsic to model's probability function $p(.)$. The challenge now is to define this function either top-down based on prior theory, or to estimate it empirically based on observed data. Due to the usually changing and progressing gameplay and the requirement for the model to work with many different games and genres, the second approach was chosen. As described in Section 3.10, the probability function is estimated on-the-fly at run time, or in other words, the model is dynamically trained on live data.

The designers' good understanding of the player and the game is a very important asset during development. But as Schell notes, it can be equally important when this understanding fails during playtesting:

> The key is to keep your eyes open for *surprises* [emphasis in the original]. To be surprised at a playtest, you must already have ideas about what will happen: players will attack level two a certain way, they will get excited at the start of level three, etc. Whenever anything out of the ordinary happens,

> good or bad, be ready to jump on it, and find a way to understand it. [Schell, 2009, p. 396]

For our stochastic models, this means that they have to match the data reasonably well, but that the situations where the model performs very poorly and mispredicts actions are not to be seen as failures, but are actually the situations where new and interesting behaviour is displayed. It is when the designer's intent or expectation and the player's actions diverge that we can gain the most insight.

3.5 Information Theory

The ideas of *expectancy* and *surprisal*, the idea that we can learn a lot from these unexpected moments described in the previous section, translate very well to information theory. Information theory is an elegant concept to quantify this surprisal and that allows us to see the user's input as abstract information that is flowing from the player to the (game) system.

Through the application of information theory on formal models of user input, we can create a formal model to measure and quantify general interaction in games (see Section 3.3). Information theory is the required tool to make the step from low-level user input to high-level interaction.

3.5.1 Information Content 101

In information theory, information content describes how much abstract information is contained in a concrete message. This is the amount of information we gain when we receive the message.

Information content, or more colloquially *surprisal*, is dependent on the message's probability. A message with lower probability, i.e., a message that is more *surprising*, carries more information than a message that is less surprising. Consider the following illustrative example:

Example Let's assume that a friend tells you that they are awaiting a baby. Depending on the person, this might be more or less unexpected, but now assume that this friend is actually telling you that they are awaiting twins. The probability of the first message is hard to quantify, but in any case we can assume that the second message, about awaiting twins, is even less likely and much more surprising. Translating this to information content, this means that the second message carries more information.

Put in other words, the more we anticipate an event or a message, the less information we gain from it. Something that is rather obvious provides us with little knowledge, whereas things that are unexpected can provide rich information. △

Information content is defined as:

$$I(X) = -\log(P(X)) \tag{3.1}$$

where X is a discrete random variable with probability mass function $P(X)$. The unit of the result depends on the base of the logarithm. In the case of the natural logarithm, the resulting unit is *nits* and in the case of the logarithmus dualis the resulting unit is the more common *bits*.

Example Consider a very simple data source that is only capable of transmitting a single symbol at a time from a finite alphabet, i.e., a discrete random variable $X = \{\odot, \otimes, \ominus, \oslash\}$ with a probability mass function $P(X) : P(\odot) = 0.5$, $P(\otimes) = 0.3$, $P(\ominus) = 0.15$, $P(\oslash) = 0.05$.

Depending on the concrete message we receive from this data source, the amount of information we gain varies. If we receive the relatively unsurprising $\odot$ symbol, we gain $-\log_2(0.5) = 1$ bit of information. On the other hand, if we receive the very rare and unexpected $\oslash$ symbol, we gain $-\log_2(0.05) = 4.32$ bit of information. △

3.6 Applying Information Theory

The core of this work is the application of information theory to stochastic models. We can use information content to measure the mismatch between the model and the actual user input. If the model predicts the input well, then the resulting mismatch or information content is small and vice versa.

The mismatch is simply computed as the information content of an input event based on the model's output. Given a model function $m(x_t|(x_{t-1}, x_{t-2}, ...))$ that returns a probability value for the current input x_t given the series of previous inputs $(x_{t-1}, x_{t-2}, ...)$, the mismatch I can be computed with the following formula:

$$I(x_t) = -\log_2(m(x_t|(x_{t-1}, x_{t-2}, ...))) \tag{3.2}$$

The term *mismatch* used throughout this chapter is introduced as a new term to describe the outcome of the higher-level model. Even though it is just another term for the information content and thus shares the same letter I in the above formula, the term *mismatch* gives it an intuitive meaning in the domain of modelling and predicting user input in games and is thus preferred. In the discussion in Section 5.3, I will come back to this and discuss this value in its original information theory meaning of abstract information that is flowing from the player to the game system.

Using the terms *prediction* and *mismatch* might also seem misleading at first. The model's predictions are not discrete, and the model does not predict one concrete action

that will be next, but predicts the probability of each action occurring next in the input. The mismatch is thus found in the probability value of that action.

The model-action mismatch lies in the open interval $(0, \inf)$ and unfortunately this interval cannot be constrained any further. On the lower end, situations and models can be constructed that come arbitrarily close to a mismatch of 0 and on the upper end, situations can be constructed with arbitrarily high mismatch. These extreme situations and corresponding models are not useful in practical analysis but only act as evidence for the definition of the bounds.

Example Consider a perfect game of Guitar Hero [Harmonix, 2005] for example. In this music and rhythm game, the player uses a special mimicry guitar controller to play simplified and adapted scores of popular songs. Note markers are scrolling over the screen, and the player tries to hit every note at the exact right time. The game was very popular in the mid to late 2000s, and many people perfected their skills to the point where they could repeatedly hit every note in a song. Once this level of skill and proficiency is reached, the gameplay comes very close to being completely deterministic. However, there is still the very narrow possibility that the player misses a note or deliberately presses a different button.

Games, by definition (for example [Juul, 2005, p. 36]), are based on some form of user interaction, and under the general assumption that human actions cannot be fully predicted, we see that the model can come very close to perfectly predicting player input in some situations, but cannot become fully deterministic.

This example illustrates a situation where a model with arbitrarily low mismatch can be found. To show the other extreme, a model with arbitrarily high mismatch, the same situation is assumed, but now the model is inverted.

One way to invert the model is to apply a monotonic function $f(x) : [0, 1] \to [0, 1]$ with $f(0) = 1$ and $f(1) = 0$ to the model function m, for example, $f(x) = 1 - x$ and to renormalise the result in order to obtain a probability function again:

$$\tilde{m}(x) = \frac{f(m(x))}{\sum_i f(m(x_i))} \tag{3.3}$$

where x is the current user input and $\sum_i x_i$ sums over the transformed probabilities of all possible inputs x_i. This way the new model will always assign the most likely event the lowest probability and vice versa. Hence, a perfect game results in arbitrarily high mismatch.

△

3.7 Model Error, Mismatch and Abstract Templates

The observable and measurable mismatch between the model and the actual user input is a combination of the intrinsic modelling error and the extrinsic mismatch caused by unexpected user interaction. In some cases like the simple gambling game of tossing a coin, the player's choice of picking heads or tails cannot be predicted and thus even the best model would always have a large mismatch.

Even though the used models have an unquantifiable modelling error and certain game situations can imply a higher lower bound on the model-action mismatch, we can still use the mismatch as a relative value in the study and evaluation of games. As long as the mismatch does not remain constant, we can use its changes to derive information from the model.

This mismatch can further be interpreted in two ways: In the first case, the model aims to minimize the model error by modelling the real-world domain it covers as closely as possible. In the second case, however, the model is regarded as an abstract template that only covers a specific aspect and does not try to model the whole domain. This template can then be used to quantify the similarity of the data with the abstract template and allows us to completely disregard the intrinsic model error because we are only interested in how well the data fits the model and not how well the model fits the data.

Example Consider a simple jump-and-run game like *Super Mario*, where we want to find a model for the underlying and *real* interaction patterns of the player. A model can be constructed with the goal to describe the interaction as good as possible, thus reducing the modelling error. The model can then be verified with data from the play session.

On the other hand, consider an abstract template that represents a certain input pattern, e.g., a special button sequence in the game. This very specific model is not expected to cover all of the interaction, but is just used to find occurrences of this concrete pattern. The resulting information content is a measure for the similarity between the data and the abstract template. △

3.8 Modelling Play with Markov Chains

So far, only a general stochastic model was assumed that provided some form of model function $m(x)$. In this section this general stochastic model will be concretised to discrete-time, discrete-space Markov chains.

On a coarse level, there are two general approaches to modelling: *Top-down*, and *bottom-up* [Canossa, 2013]. Top-down approaches are hypothesis and knowledge driven. Models are crafted based on initial observations, data or theories, which are then implemented and tested. Most models about player motivation, play experience or

emotional valence, for example, fall into this category. Bottom-up models, on the other hand, emerge from the data using post hoc analysis. Unsupervised learning techniques like clustering are an example for this group.

In between the two are hybrid models [Yannakakis et al., 2013], which incorporate approaches from both groups. The proposed modelling approach described below falls into this group. Even though the proposed model is largely based (trained) on actual data, some parameters and the approach itself are informed by previous knowledge, theories and assumptions in and about the domain.

The model class chosen for this work are discrete-time, discrete-space Markov chains. This class of models has been chosen in a top-down approach. The intention was to model play by modelling user input at a low-level so that the models remain applicable and general enough to capture a large class of games. Under this requirement, choosing raw game-controller input as the model domain was a logical consequence.

Markov chains only require a small set of a priori assumptions and parameters. These include the order of the Markov model and the discretisation approach used in the time and space domain. The remaining parameters, the entries in the transition matrix, are estimated bottom-up at run time based on observed data. Markov chains are a rather simple concept and the resulting model is not a black box, but can be easily visualised for further analysis.

Markov chains have been chosen over other, more complex models for their simplicity and generality. On the one hand, this serves economical purposes, but on the other hand, concepts that require less assumptions are generally more favourable as argued in Occam's Razor.

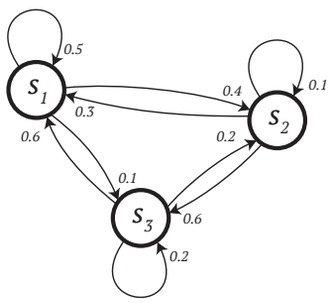
Markov chains are easily implemented, and training and evaluation of the model is computationally inexpensive and fast. They are suitable for modelling sequential patterns, and in [Lee et al., 2014], Markov chains have already been used successfully in predicting low-level user input for speculative remote rendering of fast-paced action games.

3.8.1 Markov Chains

Markov chains are a formal stochastic model that consists of a set of of n unique states $S = s_1, s_2, .., s_n$ and an $n \times n$ transition matrix A. The entries in this matrix are the transition probabilities from the current state (the rows) to the subsequent states (the columns). Figure 3.2 shows a graph visualisation of an example Markov chain.

For example, starting out from state s_1, the transition probability of transitioning to s_2 is 0.4. From there, the probability of transitioning to s_3 is 0.6, and so on. The transition probabilities of all subsequent states always has to add up to 1, or in other terms, the row-sum of the matrix is always 1.

Markov chains build on the Markov property, which states that the probability of the next transition only depends on the current state, and not on the series of all previous



	s_1	s_2	s_3
s_1	0.5	0.4	0.1
s_2	0.3	0.1	0.6
s_3	0.6	0.2	0.2

Figure 3.2: A Markov chain visualised as a graph with the transition matrix A in table form. E.g., the transition probability from s_2 to s_3 is 0.6

states that lead to it. This property is formulated as the following equality:

$$p(x_{t+1}|(x_t, x_{t-1}, x_{t-2}, ...)) = p(x_{t+1}|p_t) \tag{3.4}$$

In this equation, $p(x_{t+1}|(x_t, x_{t-1}, ...))$ is the transition probability of transitioning from the current state x_t to the next state x_{t+1} based on the sequence of all previous states $(x_t, x_{t-1}, ...)$. The Markov property now expresses that only the current state is relevant for the transition probability and thus all previous states can be ignored.

In higher-order models of order m, the Markov property is extended so that the transition matrix depends on the m previous states, i.e., (indices have been shifted for better readability)

$$p(x_t|(x_{t-1}, x_{t-2}, ...)) = p(x_t|x_{t-1}, x_{t-2}, ..x_{t-m}) \tag{3.5}$$

The Markov property is very important and makes this modelling approach feasible for modelling user input. A test session can potentially run for hours, and at a high sampling rate of for example 10 Hz, a very large number of previous states is accumulated that would have to be considered if this property was not present. The algorithmic complexity of the model is thus in $O(1)$ and not in $O(f(n))$, where n is the number of data samples.

3.9 Choosing Model Parameters

As mentioned in the beginning, this work pursues a hybrid modelling approach. Some parameters of discrete-time, discrete-space Markov chains can be selected top-down beforehand on the basis of initial assumptions and theories, and some parameters can be dynamically chosen or trained through data. The term *static parameters* will be used for the first and the term *dynamic parameters* for the latter.

The first static parameter that was chosen was the order of the Markov model. The Markov chains have been chosen to be of first order, due to the higher computational complexity of higher-order models. First-order models are also used in [Lee et al., 2014].

Other static parameters that are chosen top-down are the discretisation scheme in the time and space domain. The spatial domain in our Markov models is the input space of the game controller. The analogue triggers and the analogue sticks are discretised into approximately evenly sized intervals and cells (see Figure 3.3).

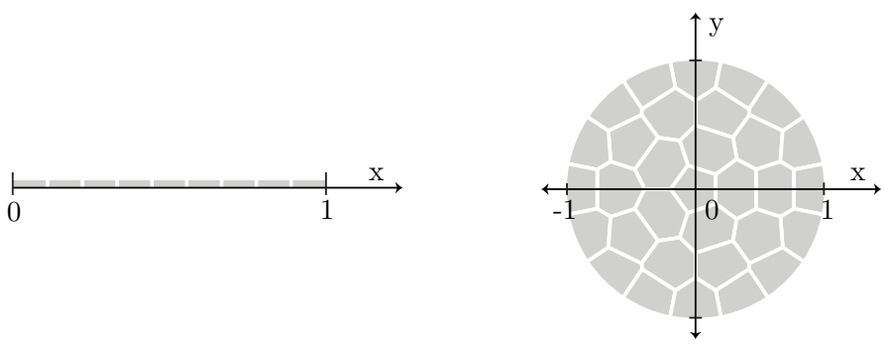


Figure 3.3: The discretisation scheme for the 1D linear and 2D circular analogue controls. The space is partitioned into approximately evenly sized intervals and cells.

This discretisation of the input space is required to uniquely map each discrete interval or cell of the 1D or 2D inputs to a state in the state space of the Markov chain. Each interval and cell is assigned a unique id, which is then used to identify the state in the Markov chain.

The Markov chain itself does not preserve the structure of the underlying space, and only regards it as a set of interconnected states. It is agnostic about the space and the discretisation approach, which allows us to freely chose them in accordance with the requirements of our domain.

A Voronoi tessellation[2] of the space has been used for the the 2D analogue sticks. The seed points for the tessellation are positioned in a regular, concentric ring pattern. This pattern, as illustrated in Figure 3.3, yields approximately evenly sized cells that are evenly distributed over the whole 2D circle that is the input space of the 2D analogue sticks.

In the current setup, analogue sticks are discretised into 32 cells each, and the analogue triggers are discretised into 9 intervals each.

The n analogue buttons are already discrete, but are organised in a binary n-dimensional space $[0, 1]^n$. The result is a binary input vector $x = (x_i, x_2, .., x_n)$, where x_i represents the current state of button i. From this vector a unique state id is generated

[2]The following JavaScript Voronoi library was used:
`https://github.com/gorhill/Javascript-Voronoi`

by transforming it into a binary string by concatenating the elements of the vector. An example vector $x' = (0, 0, 1, 0, 1)$ thus becomes the id string `00101`.

The time domain is discretised by regularly sampling at a frequency of 10Hz. This value has shown to give good results and it is also the same sampling frequency that was used in [Guardini and Maninetti, 2013].

3.10 Dynamic Modelling and Training

One of the main challenges in the modelling approach described here is to define the model function $m(.)$, or in our specific case of the Markov chains, to define the concrete values in the transition matrix A.

Using a hybrid approach, the transition matrix is considered to by dynamic in the meaning described in the previous section. Its values are not chosen top-down, but are derived empirically from the actual and individual data that is observed during each test session. This leads to the problem of estimating the transition matrix, a task that can be challenging if only little data is available as we will see in this section.

The first challenge is to define the data basis that is used to estimate the transition matrix. Most games feature multiple different gameplay situations and a general progression throughout the whole game. Additionally, players change their interaction throughout the course of a game or play session, for instance through routinisation (see Section 4.3).

This leads to the assumption that players' interaction patterns change throughout the game which makes using all available data from a play session impractical. By using all available data, the resulting model would, in the worst case, represent an average interaction pattern that is the average of many different individual and nuanced patterns.

On the other hand, games often feature similar and repeating game play segments and have only a hand full of core mechanics that dominate each segment of the game. This leads to the assumption that the intended interaction patterns in games change *slowly* most of the time. Building on this assumption, it can be argued that using only data from the last n seconds in the game can be a good approach that yields models that are agile enough to cover changing gameplay situations quickly, but that are also general enough to capture sudden and unexpected changes in player behaviour.

In the plugin this is implemented by using Markov chains of finite *length*. Length in this context means that a model is only based on data from the last n seconds. The plugin currently uses models with a length of 30 to 300 seconds. The finite length of the Markov chains together with the low computational complexity allow us to train and *untrain* models at the same time.

Training Markov chains means estimating their transition Matrix A based on data. In the naive case, the empirical probability or relative frequency can be used as the (maximum likelihood) estimator for the transition probabilities in the matrix. It is

computed by simply counting all occurrences of a given transition and then dividing it by the sum of transitions with the same starting state (cf. [Murphy, 2002]).

For dynamic *un*training, all transitions are temporarily stored, and once they are older than the defined model length, the data is removed again from the model by simply reducing the count of that transition in the matrix and adjusting the normalisation constant for that row. The following example will illustrate this:

Example Assume a system with 3 unique states A, B, C and an input series $X = (A \to A \to B \to C \to C \to B \to A \to A \to A \to C \to A \to B)$. Using the empirical probability, as described above, yields the following transition matrix. The following table shows the transition matrix for the original data set and the transition matrix where the first transition has been removed ($\tilde{X} = (A \to B \to C \to C \to B \to A \to A \to A \to C \to A \to B)$) (Table 3.3).

	A	B	C
A	$3/6$	$2/6$	$1/6$
B	$1/2$	$0/2$	$1/2$
C	$1/3$	$1/3$	$1/3$

(a)

	A	B	C
A	$\mathbf{2/5}$	$\mathbf{2/5}$	$\mathbf{1/5}$
B	$1/2$	$0/2$	$1/2$
C	$1/3$	$1/3$	$1/3$

(b)

Table 3.3: a) The transition matrix for the input series $X = (A \to A \to B \to C \to C \to B \to A \to A \to A \to C \to A \to B)$ b) the transition matrix with the first transition removed (differences highlighted in bold font).

△

3.11 Training Probability Distributions

Querying the probability value for a transition in a Markov chain is as simple as looking up the corresponding value in the transition matrix. A problem, however, that is also visible in the example above, is that the estimation of the transition matrix can suffer severe undersampling problems. For example, if we consider a 30-second Markov chain for an analogue stick discretised into 32 cells, our data basis (sampled at 10 Hz) consists of only 300 samples, i.e., an average of 9.375 transitions per cell to estimate 32 transition probabilities. This means that for such extremely short models, many cases will arise where the transition probability is either 0 or 1 and thus the information content of that transition is also either ∞ or 0.

In practice, however, when the tool is used to analyse real games, we can see that these extreme cases arise much more rarely. User input tends to be not uniformly distributed over all possible input states, and some transitions happen more often than others, as can be seen through the model exploration features in the plugin. Consequently, this

also means that more new samples fall into the well covered regions where enough data is available to estimate the transition probabilities.

In the remaining extreme cases, invalid data is clearly marked in the visualisation of the information content. Even though no information value can be associated with them, they serve as a good hint for generally *unexpected* behaviour.

In the cases where there is enough data available to estimate the transition probability, we still have too little data available to make *good* estimates most of the time. Even though the maximum likelihood estimator returns the most likely probability value, this single scalar value does not capture the uncertainty associated with it. For example, a simple 2-state model based on the data $X = (A \to A \to B \to A \to A \to B)$ has the same transition matrix as a model that is based on much more data $\tilde{X} = (A \to A \to A \to B \to A \to B \to B \to A \to A \to A \to A \to B)$, even though we can have much higher confidence in this estimate.

For the following sections, α describes the number of occurrences of a concrete transition $A \to B$ and n the number of all transitions starting at that state $A \to s_i$, where $s_i \in S \setminus B$ and $\beta = n - \alpha$.

Example Using the example data from above with $\tilde{X} = (A \to A \to A \to B \to A \to B \to B \to A \to A \to A \to A \to B)$, for the concrete transition $A \to B$ the resulting values are $\alpha = 3$, $n = 8$ and $\beta = 8 - 3 = 5$. △

To incorporate the amount of confidence in our estimate into the final model-action mismatch value, we not only estimate the scalar transition probability p with the scalar maximum likelihood estimator $p = \frac{\alpha}{\alpha+\beta}$, but instead use its full probability distribution (cf. [Murphy, 2002]). The probability distribution of p is computed using Bayesian inference by computing the a posteriori probability of the likelihood function with a neutral prior (see [Heckerman, 1996] for a good introduction into Bayesian Networks).

Bayesian inference aims at incorporating prior knowledge or information into the computation of an a posteriori probability distribution given some data. The challenge, thus, is to find a mathematical description of this prior knowledge especially in situations where no prior data or information is available. These situations suggest the use of *uninformative priors*[3] that model the fact that we do not know anything about the parameter we are trying to estimate.

One suggested distribution for an uninformative prior is the uniform distribution. Before any data is observed, it expresses that the parameter is equally likely to take any value in the given interval. For our transition probabilities this means that all transitions are equally likely to happen, all transitions are possible and all states are reachable.

The beta distribution is a reasonable choice for an uninformative prior in our case.

[3]see `https://en.wikipedia.org/wiki/Prior_probability#Uninformative_priors` for a practical introduction

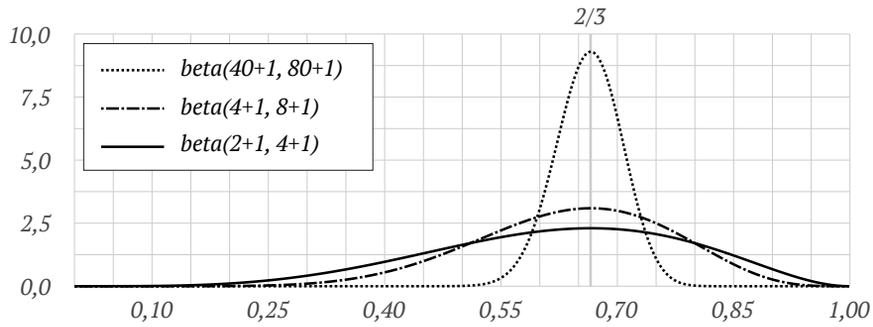


Figure 3.4: Example probability distributions of the unknown transition probability p. The more samples are available ($n = alpha + beta$), the higher the confidence in the parameter estimate.

The $beta(1, 1)$ distribution equals the uniform distribution mentioned above, and the beta distribution is the conjugate prior to the family of Bernoulli distributions, which is the distribution of our underlying problem as we will see in the following proof. Conjugate prior means that the resulting a posteriori distribution will also be a beta distribution, which makes working with it very convenient. By using Bayes' theorem we will see that the resulting a posteriori probability distribution of a transition probability is $beta(\alpha + 1, \beta + 1)$.

Figure 3.4 shows plots of example distributions. What we can see in this figure is that the uncertainty in our estimate, the width of the bump, decreases with the rising number of samples and available information.

Proof The problem of estimating the transition probabilities can be mapped to the binary problem of observing a given transition. In this case we are interested in the number of successes (α) and the number of failures (β) in our sample and transitions are assumed to be independent and identically distributed Bernoulli trials. The resulting sample probability is then given by the Binomial distribution $\binom{\alpha+\beta}{\alpha} x^\alpha (1-x)^\beta$. To get the a posteriori probability distribution $\tilde{P}(x; \alpha, \beta)$, Bayes' theorem is used:

$$\tilde{P}(x; \alpha, \beta) = \frac{\mathcal{L}(x; \alpha, \beta) P(x)}{\int_0^1 \mathcal{L}(x; \alpha, \beta) P(x) dx} \tag{3.6}$$

$$= \frac{\mathcal{L}(x; \alpha, \beta) beta(x; 1, 1)}{\int_0^1 \mathcal{L}(x; \alpha, \beta) beta(x; 1, 1) dx} \tag{3.7}$$

$$= \frac{\binom{\alpha+\beta}{\alpha} x^\alpha (1-x)^\beta beta(x; 1, 1)}{\int_0^1 \binom{\alpha+\beta}{\alpha} x^\alpha (1-x)^\beta beta(x; 1, 1) dx} \tag{3.8}$$

$$= \frac{\frac{\binom{\alpha+\beta}{\alpha} x^{\alpha}(1-x)^{\beta}}{B(1,1)}}{\int_0^1 \frac{\binom{\alpha+\beta}{\alpha} x^{\alpha}(1-x)^{\beta} dx}{B(1,1)}} \tag{3.9}$$

$$= \frac{x^{\alpha}(1-x)^{\beta}}{\int_0^1 x^{\alpha}(1-x)^{\beta} dx} \tag{3.10}$$

$$= \frac{x^{\alpha}(1-x)^{\beta}}{B(\alpha,\beta)} \tag{3.11}$$

$$= beta(x; \alpha+1, \beta+1) \tag{3.12}$$

where $\mathcal{L}(.)$ is the likelihood function and $P(.)$ is the prior probability function. The pdf of the beta distribution is $beta(\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$ and the beta function is defined as $B(\alpha,\beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1}dx$. In Equation (3.10), the constant factors are reduced from the fraction. □

Using the a posteriori probability distribution of our parameter, the transition probability, we can now incorporate the confidence we have in our estimate into the computation of the model-action mismatch using the information content. The only problem is that the formula for information content is only defined for scalar probability values, not for probability distribution functions.

Instead, we have to use *differential entropy*, a generalisation of information content (cf. [Cover and Thomas, 1991, p. 224]). This generalisation allows us to compute the information content (more precisely the entropy, as the name suggests) of the probability distribution of our transition probability. Differential entropy is defined as:

$$h(X) = -\int_{x \in S} f(x) \log(f(x)) dx \tag{3.13}$$

where X is the random variable, $f(x)$ is the probability density function and S is the support set of the random variable. In the implementation of the plugin, this integral is solved using numerical integration and the results are cached for better performance.

Especially in situations where only few samples are available and thus α and β are small, using the differential entropy gives less extreme values.

3.12 Implementation

To perform the prestudy in Section 4, the model was implemented in the *Information Flow* plugin for real-time analysis of raw user input in games. Figure 3.5 shows a screenshot of the final Space Walk plugin in use. This plugin allows a number of different Markov

chains to be trained on-the-fly in real time and also to be used for analysis at the same time.

Figure 3.5: A screenshot of the *Information Flow* plugin. It shows a) general and model specific settings b) the visualisations of the model-action mismatch c) visualisation settings for the exploratory model visualisation d) extensive visualisation for linear and circular analogue inputs.

The intended use case for this plugin is the playtesting use case described in Section 2.2. In this context, a designer or developer can use the plugin during the testing session as an aggregate data source that captures many aspects of play in one single plot. That way, a designer can quickly see if the player's interaction patterns change, an indicator for design issues in the game.

The input data for the plugin is created with an additional external tool that samples game-controller input at regular intervals of 10 Hz. The standard controller layout assumed in this plugin is shown in Figure 3.6. It consists of two 2-axis analogue sticks located underneath either thumb, one analogue trigger for each index finger, and a number of digital buttons. The directional pad to the left is treated like separate buttons without any special interconnection between them.

The plugin was implemented in JavaScript and uses the D3.js library [Bostock et al., 2011] for the real-time visualisation of the results. JavaScript and the Space Walk framework allowed for quick iterations and fast prototyping of the various features of this plugin. JavaScript also offers a good ecosystem of open source libraries, for instance for mathematical and geometric functions. Even though JavaScript is not the highest performing option when it comes to real-time data visualisation, it has proven to be fast enough for the requirements of this work.

3.12.1 User Interface

The plugin dynamically trains the model at run time (see following Section 3.10) and visualises the resulting model-action mismatch of the input as a rolling bar plot and also

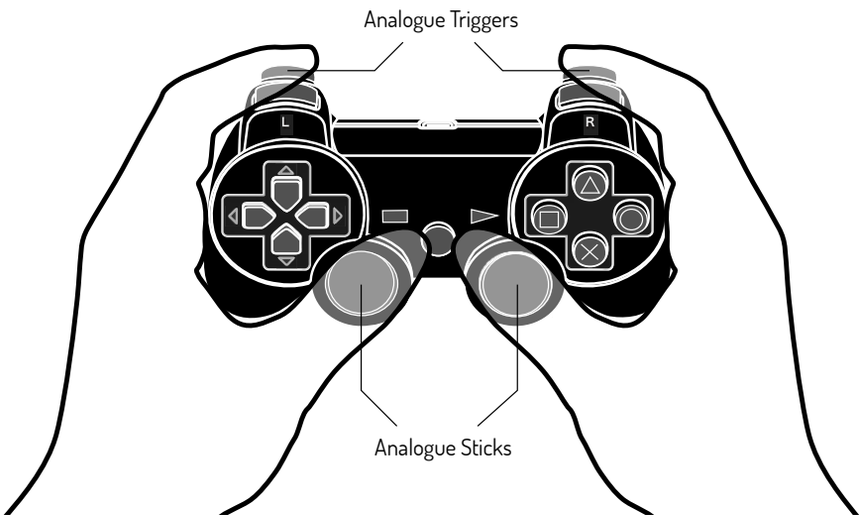


Figure 3.6: The generic controller layout assumed for this plugin.

offers extensive model-exploration features. The user interface offers a number of controls to manage general settings like the used game controllers and their mapping as well as setting model parameters and loading and exporting model data.

The model-action mismatch plot (see Figure 3.7) shows a comparative visualisation of two independent models. Parameters for the models A and B can be chosen individually, and the resulting data is visualised in a split view. The upper half represents data from model A and the lower half from model B. Differences in the data are highlighted by brighter bar segments that visualise how much a value in one model is larger than the corresponding value in the other model.

Additionally to comparing data from the two models, the results are split into the contributions from analogue and digital inputs, and the topmost plot shows the combination of the two.

The plugin also offers extensive visualisation features to further explore some aspects of the data. Heatmaps of the analogue stick positions and vector-flow visualisations of the stick movements are available and subsequent stick positions are shown for each state in a matrix plot.

3.12.2 Leveraging the Additivity of Information Content

Even though Markov models are computationally simple for smaller models, the size of the $n \times n$ transition matrix A increases relative to the square of the number of unique states n in the model. There are currently 32 states for each analogue stick and 9 for each analogue trigger, plus an additionally large number of digital button states. Modelling all these in a single model can be infeasible, especially since the plugin is developed for the web browser platform.

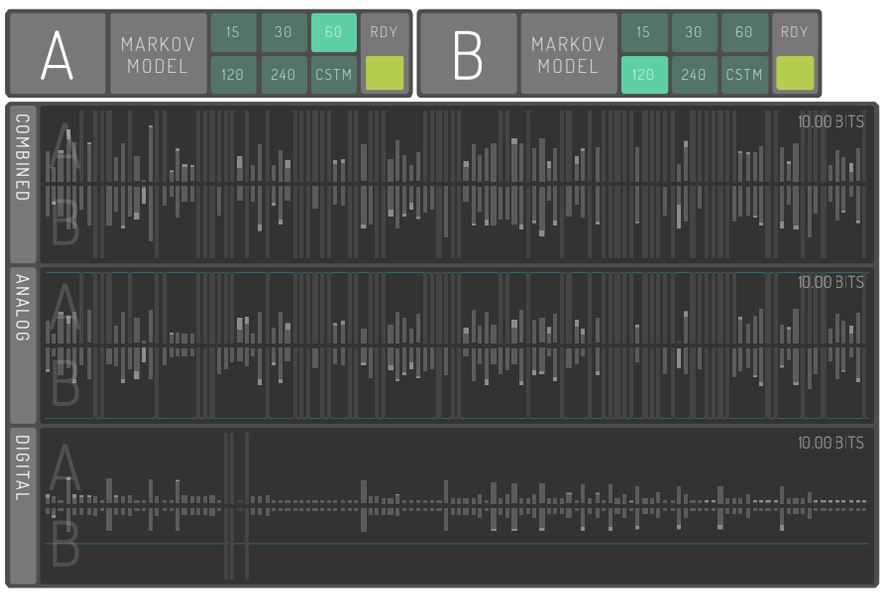


Figure 3.7: The visualisation of the model-action mismatch and controls to set the length of the two models (A and B) used for the comparative visualisation.

Luckily, the additivity of information content can be leveraged to mitigate this problem and to reduce the size of the transition matrix. Additivity of information content expresses that for an event C that is the intersection of two stochastically independent events A and B, the information content of C is the sum of the information content of A and B.

$$
\begin{aligned}
&\text{let } A, B \text{ stoch. independent}, C = A \cap B \\
I(C) &= -\log(p(C)) \\
&= -\log(p(A) \cdot p(B)) \\
&= -(\log(p(A)) + \log(p(B)) \\
&= I(A) + I(B) \quad \square
\end{aligned}
$$

This allows us to model analogue sticks, triggers and buttons independently under the assumption that these inputs are stochastically independent of each other. Many games use triggers, sticks and buttons independently of each other to control different elements in the game that are not or only loosely coupled. The independent usage of the different inputs is thus a strong hint that the stochastical dependence is weak in most cases. The assumption of stochastical independence does of course not hold generally and needs to be further explored, but it is used as a justifiable modelling assumption.

For the large space of discrete input states (somewhere in the range of 2^{16} depending on the number of buttons on the controller), a sparse transition matrix is used, which

only covers the transitions that have actually been observed in the data. This reduces the size of the matrix drastically to the number of unique button states and transitions that actually occur in a game session. The practical tests showed that most of the time only single buttons are pressed and that n-button combinations are increasingly rarer with rising n.

CHAPTER 4

Results

An exploratory prestudy was performed analysing a number of commercially released or in development games to see what patterns emerge from the data and to see how the quantification of interaction in games performs under real-world conditions. This prestudy is the first step to test and verify the underlying assumptions and hypotheses.

The following sections discuss the setup and the methodology of the study. Preliminary results are presented, and in Section 4.3 additional findings in the field of routinisation are highlighted that directly build on the model and the results from this thesis.

4.1 Methodology

A number of games were analysed with the Information Flow plugin, and the results of the test sessions were recorded in a combined video stream that was then analysed to find patterns and structure in the data.

The games used in the pre study ranged from racing (Drift Stage[1], Beyond 35000[2]) to AAA first person action games (BioShock[3], Spec Ops: The Line [4], Portal 2 [5]) to 2D action games (Secrets of Rætikon[6], Super Meat Boy[7], Braid[8]) to a first person exploration game (Proteus[9]). The test sessions lasted between 6 and 40 minutes and were performed on a regular gaming PC with a 24-inch monitor sitting on the desktop in an uncontrolled environment.

[1] http://www.driftstagegame.com
[2] http://beyond35000.com/
[3] https://en.wikipedia.org/wiki/BioShock
[4] https://en.wikipedia.org/wiki/Spec_Ops:_The_Line
[5] https://en.wikipedia.org/wiki/Portal_2
[6] https://en.wikipedia.org/wiki/Secrets_of_Rætikon
[7] https://en.wikipedia.org/wiki/Super_Meat_Boy
[8] https://en.wikipedia.org/wiki/Braid_(video_game)
[9] https://en.wikipedia.org/wiki/Proteus_(video_game)

The tool output and additional data were captured during the test using the Open Broadcasting Software (OBS)[10]. Multiple video streams were spliced together into a single video that was then used as the basis for later analysis. Figure 4.1 shows a frame from the resulting video. It includes the video feed from the game, a direct visualisation of the controller input, a video feed of the player, and a capture of the Information Flow plugin used. Using OBS, the video streams are combined during recording, which takes away the need of syncing the individual streams afterwards.

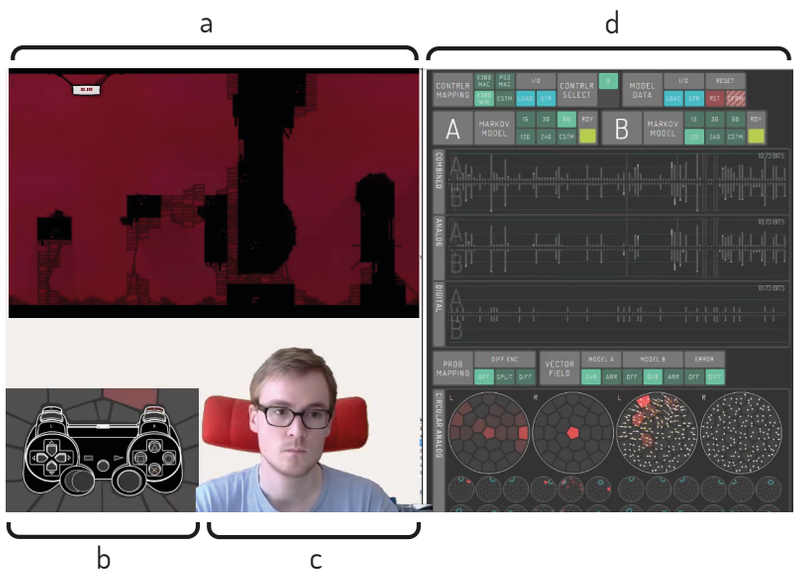


Figure 4.1: A frame from the resulting video used during the analysis. Several video streams have been merged together: a) the video feed from the game b) the direct visualisation of the controller input c) a video feed of the player d) output of the Information Flow plugin.

The resulting videos (one for each test session) were screened to find patterns or other points of interest in the videos. Various patterns could be identified and possible explanations for these patterns are given in the following section.

The figures shown in the results section are distilled from the video recordings of the test sessions. The figures highlight a representative and exemplary occurrence of the pattern in the game. Figure 4.2 serves as a guidance on how to interpret the following figures. The information content of each tick (regular time interval) is shown in the horizontal bar chart. The horizontal dimension displays the time and the vertical dimension displays the information content measured in bits in the linear range from 0 to 10 bit.

Due to the problem of not having enough data available to estimate all model parameters, the situation can occur that a new input event is encountered that was not

[10] `https://obsproject.com`

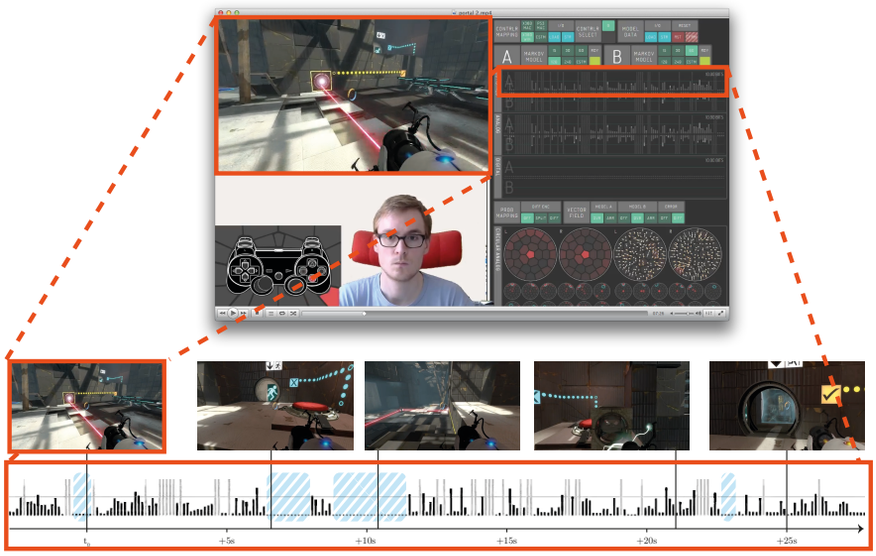


Figure 4.2: The results figures explained: The bar chart on the bottom shows the information content for each sequential action and the screenshots above illustrate the game context.

encountered in the data that is used to train the model. These input events are thus *infinitely surprising*, which is shown as a grey bar of full height in the bar plot. Even though no concrete information content can be attributed to these events, they are very indicative of generally unexpected behaviour and are thus very valuable in the data analysis.

In addition to the information content plot, frames from the game are provided to better illustrate the course of action and progression in the game situations. The individual frames are linked to the corresponding time in the plot.

4.2 Preliminary Results

A number of different patterns could be observed in the various games. The results from this prestudy are the first steps into a new direction of objectively measuring interaction in games and identifying the underlying patterns. The preliminary results presented here are to be seen as indicative and have to be further validated in future studies.

Not all games showed patterns that could be identified as such, or only showed a very weak indication for different patterns. These games were *Bioshock 2*, *Spec Ops: The Line* and *Secrets of Rætikon*. The gameplay in these games is multifaceted and more complex. Additional data needs to be gathered to see if and what patterns emerge from these games.

An expected feature of the found patterns is that they are relatively short in relation to the length of the model that was used during the test. This makes intuitively sense,

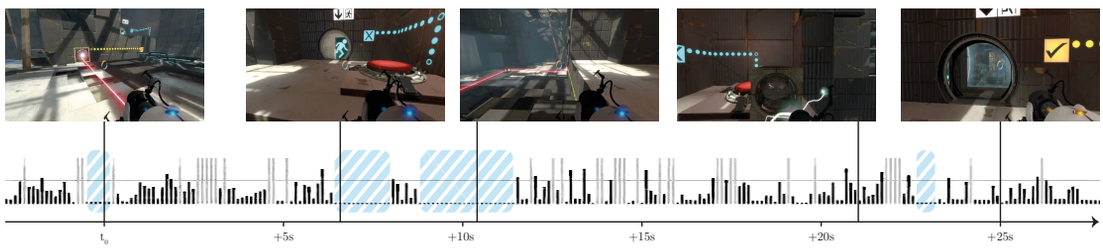


Figure 4.3: Results from the game *Portal 2*: The action breakdowns can clearly be seen in the data as phases of no or very little interaction highlighted in blue. (Model length = 120s)

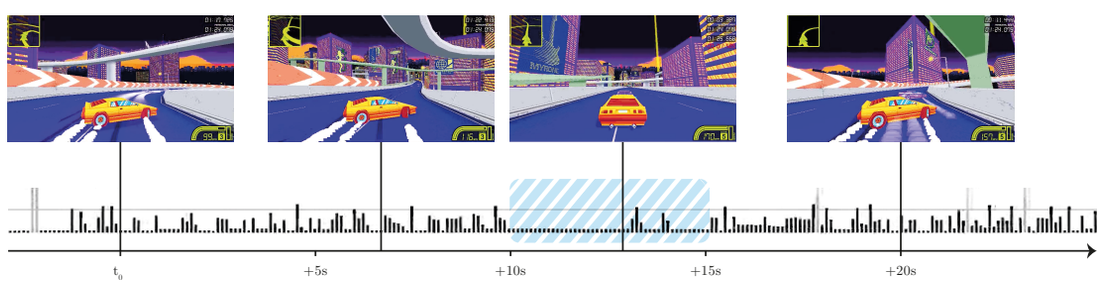


Figure 4.4: Results from the game *Drift Stage*: The game shows continuously high levels of interaction except for segments that are relatively easy track segments like straights highlighted in blue. (Model length = 120s)

since the model can only detect *sudden* changes in the behaviour. If the rate of change in player behaviour would approach the rate of adaptation or learning of the models, than the resulting model output would be inconclusive.

The game *Portal 2* showed strong indication of a pattern that appears very similar to the *action breakdown* pattern described in [Gamberini et al., 2011]. In this game, the player tries to unlock the exit of a room by solving puzzles in the room. The pattern that could be observed in the data as intervals of little to no information content (Figure 4.3) was that the player repeatedly stopped all interaction for brief moments and then continued to solve the puzzle. Using action breakdowns as a explanation, it appears that the player performs an action until they don't know how to proceed further. They stop, reorient themselves, form a new plan and then continue onwards.

The racing game *Drift Stage* showed continuously high levels of information content (Figure 4.4). This fast-paced racing game requires constant interaction to correct the car's trajectory to keep it on the track and to avoid crashing. It is only in *easy* sections like straights that the information content drops. The other tested racing game, *Beyond 35000* however, did not show these phases of lower information content (Figure 4.5). This could be explained by the different game mechanics of the game. In this game, a flying vehicle is controlled in a fully 3D space, which is more challenging, and even in straight sections the vehicle's up and down path has to be adjusted permanently.

The puzzle platformer *Braid* showed an almost discretised interaction pattern even though the game mechanics are not discrete (Figure 4.6). The level design, however,

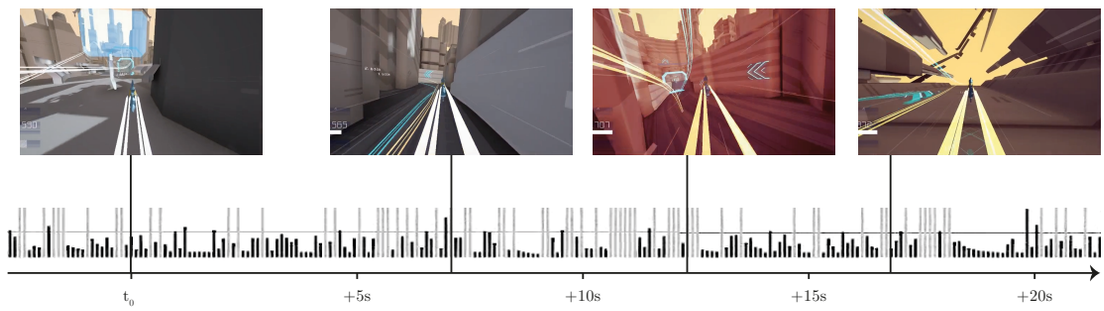


Figure 4.5: Results from the game *Beyond 35000*: Contrary to the game *Drift stage*, interaction levels are always high, because even in straight sections, the ship's up and down path has to be adjusted permanently. (Model length = 60s)

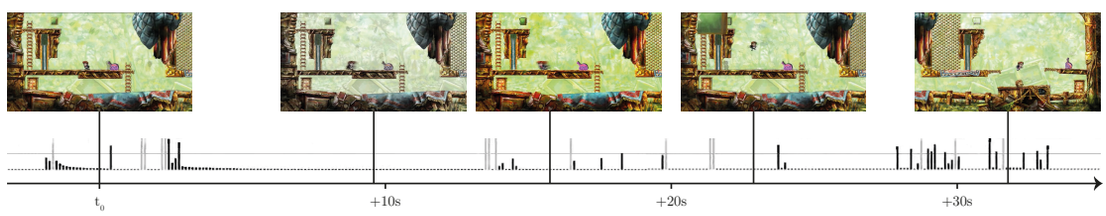


Figure 4.6: Results from the game *Braid*: Even though the game space is continuous, the data interaction appears to be almost discretised, showing only spikes of interaction. (Model length = 120s)

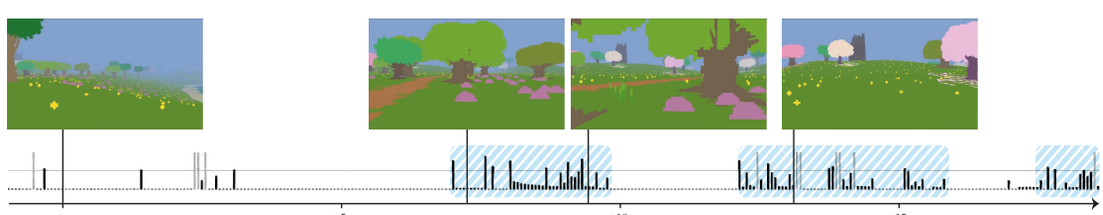


Figure 4.7: Results from the game *Proteus*: This slow-paced first person game shows generally low levels of interaction, except for phases when the player actively looks around, or interacts with the environment, highlighted in blue. (Model length = 120s)

seems to induce a discretisation of the game space, which showed up as very short bursts of interaction in the data.

The first-person exploration game *Proteus* showed a similar pattern as Braid (Figure 4.7) but due to the different nature of the game, a different explanation for the pattern seems more likely. *Proteus* has very relaxed gameplay, and the pattern seems to indicate that the player picks a target destination and then walks straight towards it, resulting in very little interaction. Interaction only increases when the player changes their plan, interacts with the environment or looks around. This pattern seems to almost be the opposite of action breakdowns.

The results I want to highlight here, are from the 2D action platformer game *Super*

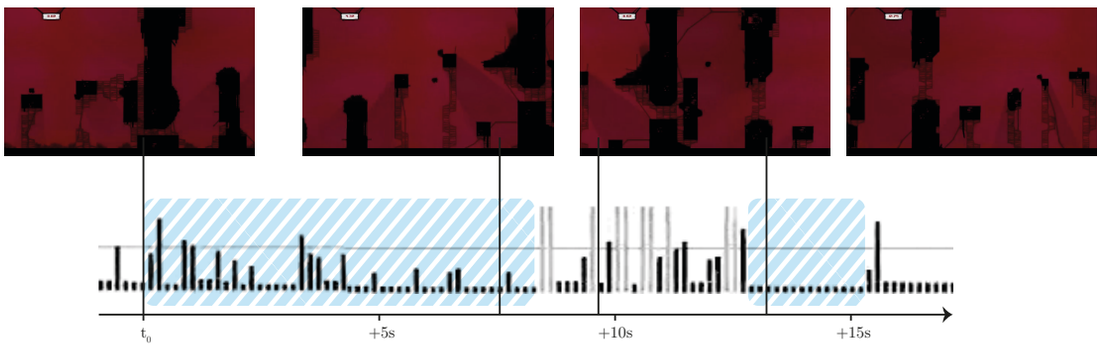


Figure 4.8: Results from the game *Super Meat Boy*: Routinisation of the player could be observed the micro level within individual level segments. The plot highlights sections where the player is routinised as opposed to sections where the player is not. (Model length = 60s)

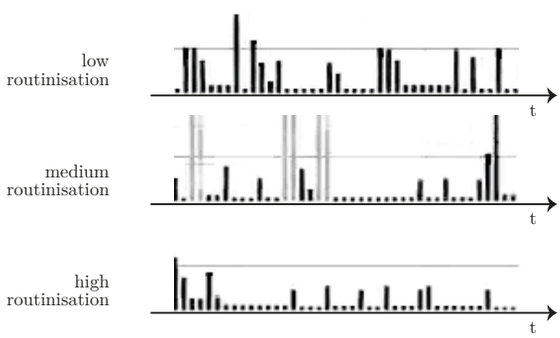


Figure 4.9: Information content plots from the beginning of a level of *Super Meat Boy*. Each plot shows data from the same level segment but at progressively higher levels of routinisation that illustrate the overall falling information content of the interaction.

Meat Boy. In this game the player tries to overcome many deadly obstacles in a level to reach the exit. A successful playthrough of a level only takes a few seconds, but it usually takes multiple tries before the level can be mastered. The game was specifically designed for this mechanic, and as an immediate reward for the player, all attempts of the level are overlayed and shown in a single replay upon the completion of the level.

In this game, the players become more and more routinised, up to the point where they master a level, a pattern that could be observed through falling information content levels throughout this process (see Figure 4.9). On a micro level, a similar pattern could also be observed for specific level segments, in which the player appeared more proficient than in others that posed a bigger challenge (Figure 4.8).

The process of the player improving their play during a level or play session can be described by *routinisation*, a concept from the field of action and practice theory (see next section). Based on the data from this preliminary study, routinisation in games has been further explored, and the results are summarised in the following section.

4.3 Routinisation in Games

The work on routinisation in games builds on the tools and models created in this thesis and on data from the prestudy. It was performed in cooperation with Martin Pichlmair from the IT University of Copenhagen and a works-in-progress abstract is to be published in the proceedings of the ACM SIGCHI Annual Symposium on Computer-Human Interaction in Play (CHI PLAY) [Wallner et al., 2015]. The following sections provide a summary of the work and are meant as an excursion.

4.3.1 Defining Routinisation

Many processes in our everyday lives are performed without conscious effort. This internalisation of an activity has been described as routinisation, and even though routinisation is well covered in the context of work [Yamagata-Lynch, 2010; Nicolini, 2012] and human-computer interaction [Kaptelinin, 1995], it has not been applied to games yet.

Leont'ev [1974] illustrates routinisation with the example of manual gear-shifting while driving a car. When learning to drive, shifting gears is a conscious process with an explicit goal. Later it becomes routine and

> "can no longer be picked out as a special goal-directed process: its goal is not picked out and discerned by the driver; and for the driver, gear shifting psychologically ceases to exist" [Leont'ev, 1974] in [Nardi, 1995].

This implies that the immediate motoric interaction is getting more and more independent of the conscious, goal-oriented interaction. We also build our definition of routinisatition in games on practice theory, a scientific field originating in the social sciences [Bourdieu, 1977; Giddens, 1984], the routinization of bodily activities results in 'practices':

> "Practices are routinized bodily activities; as interconnected complexes of behavioral acts they are movements of the body. A social practice is the product of training the body in a certain way: when we learn a practice, we learn to be bodies in a certain way (and this means more than to 'use our bodies'). A practice can be understood as the regular, skilful 'performance' of (human) bodies." [Reckwitz, 2002, p. 251]

In the context of our work, we summarise the qualities of regularity and skillfulness as proficiency.

Gameplay elements are often repetitive and players practice to overcome them. In extreme cases like e-sports or speedruns[11], players become routinised to the point where they reach a very high level of proficiency and deliver virtuoso performances.

[11] A special form of play where players try to finish a game in the least amount of time.

Routinised play is individual to each player and can be viewed on different levels. When we talk about routinised play, we regard it as specific to the game and the player that is playing. Even though most players play a given section of a game quite similarly, individual players have slightly different and unique interaction patterns, just like different drivers exhibit different interaction patterns when driving the same car.

For the purpose of our research, we define the properties of routinised play as:

- routinised play is individual to each player and game
- routinised play is repeatable and self similar
- routinised play appears proficient

4.3.2 Modelling Routinisation

In the paper, we argue that routinisation can be modelled through the discrete-time, discrete-space Markov chains described above. Using our definition of routinisation in games, we can see why Markov chains suit this problem well. The self similarity of routinised play is what makes training Markov models from data feasible. If it were not self similar, the transition matrix of the model would not converge to a stable configuration during training (cf. Section 3.8).

The individuality of routinised play with respect to each combination of player, game and setting is covered by training the dynamic parameters of our models on-the-fly. By choosing the length of the model accordingly, models can be selected that adjust quicker to new situations with the trade-off of having less data available.

4.3.3 Approaching Routinisation

Routinisation in games can be seen as a gradual and continuous process where players' actions are becoming more and more routinised with practice. This means that their interaction converges to a routinised state while at the same time, the model, based on the live data, also converges to that same state. It is only in the end, when the conditions of routinised play are met, that the model matches the interaction.

It is this process that could be observed during the preliminary prestudy of the 2D action platformer game Super Meat Boy. The observed model mismatch was gradually decreasing with repeated tries of a level. The actions became more and more routinised, and the model predicted the actions better and better (see Figure 4.9)

CHAPTER 5

Discussion

The preliminary results presented showed that patterns could be found in some games that also matched simple and plausible explanations. These first results are a strong indication that the approach can generate meaningful data and that the proposed quantification and objectification of interaction in games is applicable in real-world scenarios. Additional data has to be gathered and further studies are required to extend and validate the results.

The following sections discuss a range of practical and theoretical implication of this work. First, the practical implications and possible applications of Space Walk and the results of this work are discussed. As a corollary to that, the role of Space Walk and the developed plugins is elaborated. It is shown that the unintuitive practice of not saving data for later analysis can have practical advantages in real life productions.

In the theoretical area, the information content of the user input is discussed as a general measure for interaction in games under the theoretical assumption that a perfect formal model of user input can exist. Before closing this chapter with an outlook on future work, Section 5.4 explores the fringes of games by looking at the relation of interaction and agency.

5.1 Practical Implications and Applications

The results outlined above show patterns emerging for some games. In game development, these patterns can be used as the basis to make systematic changes and adjustment to the game's design to either support or weaken certain patterns. As an example. the racing game Beyond 35000 showed continuously high interaction levels, which can be a hint that the game lacks proper pacing or that the levels don't offer enough variety. The frequency and the length of the action breakdowns observed in the game Portal 2 could

be used as a metric to analyse the level's difficulty and to ensure that the game has an overall difficulty progression over the whole game.

The formal models and the quantification of play can be used in several applications. A very obvious application is the use during game testing. Changes in the information content can be used to identify problems in games or potential points of interest (POI) in a dataset that warrant further analysis. The developed tools also allow a real-time analysis of play, which can be used during testing, to make sure no aspects of a test session are overlooked, and to give the designers a way to monitor a test session, and to react in the moment when problems occur.

Improvements in modelling and new approaches can for example be used in creating better and more human-like adaptive AI, as it has already been done in past games, for example in the racing game *Juiced 2*[1], to model opponent behaviour in asynchronous online multiplayer races.

With the current-generation game consoles, the importance of sharing in-game footage via social media rose sharply, and automatic POI extraction could be used to automatically generate highlight reels to share or to relive important and exciting moments in the games.

5.2 Ephemeral Tools

Space Walk and the developed plugins represent a special class of analysis tools: *Ephemeral tools*. Ephemeral tools produce results in real-time, but the data is *not* automatically retained for later analysis. This might seem counterintuitive at first, but there are several reasons why explicitly using ephemeral tools can be beneficial.

The benefits of using real-time tools are immediately clear to most users: They produce results immediately without lengthy processing times, parameters can be changed on-the-fly and become effective immediately, they provide immediate feedback for the user and facilitate the interaction with the test subject and setup. In the case of a playtesting session, for example, real-time tools give the designers the opportunity to immediately ask questions related to the current context or provide additional help and information about the game if needed.

Analysing data either post hoc or hypothesis driven has a certain cost associated with it. It takes time to perform the analysis, and in some situation it might also be hard to estimate the time required until results are reached. In some situations, no workable results are found within a given time. The interviews in Chapter 2 and additional colloquial evidence from other developers also suggest that in some cases, the resources needed to perform a thorough analysis are just not available. Large amounts of data are collected, just to be carefully shelved never to be revisited again.

By their nature, ephemeral tools cannot achieve what other, more sophisticated tools

[1] `https://en.wikipedia.org/wiki/Juiced_2:_Hot_Import_Nights`

can achieve, but when it comes to cost, and maybe even cost effectiveness, ephemeral tools can prove to be a valuable lower-end extension to established testing and analysis suites.

Cost in this context means the depletable and non-depletable resources needed to use the tool. Depletable resources are the time and person hours it takes to run a test, the infrastructure used as well as computational and storage resources. Non-depletable resources on the other hand are the knowledge and experience required.

Ephemeral tools ideally require only setup costs that are no higher than the setup cost of other testing practices. The practicalities of running a test are similar in many cases, and using ephemeral tools should incur little to no extra cost. Additionally, ephemeral tools can be used ad hoc, and don't require lengthy study planing or a longer data collection period.

The running cost are by definition limited to the resources needed during the test, because there is no post-test analysis phase. Additionally, the worst-case cost, the case where a tool is used that does not generate any useful insight, is also bounded by the same limit. Furthermore, in these worst-case situations, the cost can be much lower. A designer can just ignore the tool during a test and then occasionally return to it, to see if it generated useful data in the meantime.

Even though ephemeral tools are a wonderful thing on their own, the gap to other approaches, i.e., not retaining data, can be easily bridged in various ways. In the prestudy described above, an additional tool was used to capture and merge video streams of the game, the player as well as tool output into a single video that was then later used in exploratory analysis. Using this technique allowed combining the benefits of both: having the data available ad hoc during the test, but also having it available afterwards for further analysis.

Due to the low cost, ephemeral tools can be used together with other practices as a first step to get an overview over the data, as a preview for possible results or just as a control monitor during a test to see and control that all parameters are in their expected range.

In conclusion, the operational risk of using ephemeral tools is generally low, set-up cost is low, the maximum cost is strictly bounded, and in the worst case, cost is much lower than that in most cases. Many ephemeral tools are probably already in widespread use. It is those little debug helpers and visualisations, the little informal tools created to support playtesting or little status monitors or live statistics that are already embracing ephemerality.

5.3 Input as Information

Interpreting user interaction or user input in games as abstract information creates a new way of looking at games. If we assume the existence of a general model that perfectly captures user interaction, then we can interpret the resulting information content not

just as the model error or model-action mismatch, but we can then directly quantify interaction, and compare it between different games, genres, play situations and players.

Using the proposed model of interaction, which is limited to modelling only a well-defined aspect of interaction, we can also reason about interaction in games, but the results cannot be taken as absolute values. The model used in different situations, different games or genres, can have a different intrinsic model error. We can thus only directly compare the results of of similar situations.

Even though the general model that perfectly describes interaction is not available yet, we can use it as a theoretical tool to make assumptions about games. We can use it to reason about the theoretical limits of interaction, about what elements define a game, and how they relate to each other.

The question of how much information content a certain action carries is also a very important aspect of agency, the capacity of the player to act in a game. If the player's interaction does not carry any information, can it still be used to *act* in the game?

5.4 Interaction and Agency

Section 3.3 discussed the assumption that by modelling raw user input, 100 % of the user's interaction with the game can be captured. Another aspect of this assumption is that interaction is assumed to *happen* between the player and the game system, which can further be seen as information travelling from the player to the game (see previous section).

This section discusses and ultimately challenges this assumption in some situations by exploring it through two real-world examples that lie on the fringes of common game definitions. Agency plays an important role in this discussion to locate where interaction is actually *happening*.

Let's revisit the Guitar Hero [Harmonix, 2005] example, and look at it from the point of view of agency. As the players improve in the game, they hit more and more notes in the game and make less mistakes. Once they reach a certain level of proficiency, they manage to repeatedly hit all the notes, their interaction becomes almost completely predictable and the information content of the interaction thus drops to almost 0. If we were to create a test session where we could spoof the user input and thus eliminate the user's input completely, would the players still feel the same agency and sense of accomplishment, even though their actions had no effect on the game? What if they accidentally hit a wrong note, would it immediately take away their perceived agency, and how would they react?

As another example, consider a simple gambling example also mentioned in the very beginning (Section 1.3). If we are betting on a fair coin toss, our choice of heads or tails does not matter for the outcome of the game. The only extrinsic interaction we have is the choice to play or not to play. Still, however, some people can deeply engage in these simple games and experience a strong feeling of agency. If we consider it a game, but

the player's action have no real impact on the outcome, where is the interaction taking place? What we sometimes see in similar situations is that people are reaching into the realm of superstition while playing. They pray to their gods, perform rituals like blowing on the dice, or always using the exact same routine when throwing a coin in order to improve their odds. They craft theories about how many times a coin can land on heads, before it *has to be* tails the next time and so on.

In all these examples of games without interaction, or at least games with no information flowing from the player to the game system, we can still argue that (some) players still experience agency. The two possible conclusions are either that interaction is not necessary for the player to feel agency, or, more likely, that the premise that interaction is based on the information flowing from the player to the game does not hold in all situations. Instead, it seems that a lot of the interaction is actually happening *within* the players.

The preliminary conclusion thus is that the boundaries of game systems are not always that clear cut, and can also include the players and their context as well. This also strongly speaks in favour of holistic design approaches like the MDA framework Hunicke et al. [2004], which goes beyond regarding abstract rule systems in isolation.

The quantification of interaction in games provides a tool to explore and elaborate all these questions, assumptions and preliminary conclusions. Through objectively measuring interaction in games and interpreting interaction as information, we can design studies to further explore these topics.

5.5 Future Work

Future work can be roughly separated in two directions. The first direction covers work to improve on the current models of interaction and on the implementation: To explore the effect of the modelling parameters, to try new modelling approaches and to generally reduce the modelling error. The second directions focuses on the implications of this work. Given a quantification of interaction, how can we use it in game analysis and game design?

Moving in the first direction, the Information Flow plugin already contains a number of comparative visualisations to compare different models on-the-fly. These features have to be further explored and their use has to be evaluated in larger-scale user studies. There is also room for additional analysis features in the plugin, like computing change rates and derivatives of the data, to better analyse the dynamic behaviour of the information flow.

In the area of modelling, the resulting patterns have to be explored further, to create better models of play or of specific aspects of play. The assumption of stochastic independence of the input from the various buttons and analogue controls has to be reviewed, by comparing the results from models that use a single Markov chain to models that use independent Markov chains.

To better capture the dynamic properties of play, computing the probability of input chains, instead of only computing the probability of the last input, should be considered. This could make the approach more independent of the sampling rate without changing the underlying modelling structure. Currently, an action in a series of identical input always carries the same information content (if dynamic learning is ignored). This might be contrary to the expectation of a designer that input *has to change* at some point. This would translate into a rising information content, analogue to the designers rising anticipation of a certain action. Computing the probability of longer input series could solve this problem.

A parameter study should be performed to analyse the effects of the static model parameters, the sampling frequency, the variable model length and the discretisation approach in the space domain.

CHAPTER 6

Conclusion

This work represents the first step into the new field of quantifying interaction in games by applying information theory to formal models of user input. Through this work, we can reason about interaction in games by looking at the structure and patterns of a specific low-level aspect: raw user input.

Modelling raw user input with discrete-time, discrete-space Markov chains proved to be adequate both from a theoretical and practical standpoint. Further work on routinization in games supported this position by showing that the qualities of routinised play (individuality, self similarity, proficiency) are also well translated into the domain of Markov chains.

Information content also proved to be an adequate measuring tool to quantify the mismatch between the stochastic model and the actual user input. Seeing information content as *surprisal* also creates a strong connection to the design process and game testing where designers are interested in finding and eliciting unexpected player behaviour during tests that does not match their design intent.

The described modelling and measuring approaches were implemented as a plugin for a web-based framework that was created as the workpad for this research work. Even though it was implemented in JavaScript, a scripting language that is not focused on runtime performance, the web browser platform proved to be capable of the simultaneous training and evaluation of multiple Markov models as well as extensive visualisations of the models and the results in real time.

This tool was also used in an exploratory prestudy with 9 different games from various genres. Preliminary results indicate the emergence of different interaction patterns. One of the patterns is *routinisation*, which was further explored in a cooperation with Martin Pichlmair from the IT University Copenhagen [Wallner et al., 2015].

The work of this thesis also has implications on analysing games by quantifying

interaction using the model and approaches described in this thesis. In conclusion, I would like to revisit the open-ended guiding research questions outlined in my personal motivation, to see in what way this work touched on them beyond the core contributions of this thesis.

What characterises the player-game interaction? Through this work we see that the player-game interaction can be looked at by analysing the underlying structures and patterns. The analysed games showed a number of different patterns, and they also showed that play is a very individual process, which is dependent on the player, the game, as well as the context they are played in.

If we interpret interaction as the pure information exchange between the player and the game, we reach a new point of view on games that creates new angles for analysis. Looking only at the amount of information that is flowing from the player to the game (system), we can use this quantification to create a new taxonomy of games. By having a formal definition of interaction, we can furthermore analyse *where* interaction takes place.

Where does player-game interaction take place? Looking at the limiting cases of the information content in games that have little to no direct interaction, we see that interaction in games does not always happen between the player and the game system as such. I am quite sure that we can construct experimental games that completely cut the information flow from the player to the game, but that are still perceived as games by the players. This implies that the game as such is more than the input and output of a rule-based system, and also includes the players and their wider context.

How can we make it measurable? Lastly, information theory proved to be a very good tool for quantifying interaction in games. The concept of abstract information fits the concept of interaction very well, but it is still very dependent on the underlying stochastic models. As long as we do not have a perfect model of general interaction, we cannot measure absolute values of *interactivity* in games. What we can do, however, is measuring certain aspects or patterns of interactivity that are well captured by specialised models. Further work on routinisation in the next months will show if this assumption holds, but from where we are right now, the future looks very bright.

APPENDIX A

Interviews

To further dive into this field and to create a working definition of playtesting, additional interviews with game designers and developers were performed at the beginning of this work. Developers from Austria, Denmark, France, the UK and the US were interviewed to find out about the different aspects of playtesting and to see where their definitions of playtesting agree and where they differ.

In the following paragraphs I will present parts of the interviews to illustrate the different viewpoints on and the different approaches with playtesting.

Andrea Schmoll, Game Systems Designer at Arkane Studios, France:

> On my previous projects, play tests were often a good way of receiving user feedback on the game during a very early phase of development, as well as using these tests as some kind of external structure for the project. Playtests happened every three weeks on one of those projects, which meant we had a definite deadline for fixing old bugs, adding a certain amount of features to test the next time, and not running into the risk of either spending too much time on certain features, or, the opposite, not adding features for various reasons.
>
> As mentioned briefly, those play tests would take place early on during production to gather as much information on the user experience as early as possible. This way, we would be able to quickly iterate on some features and issues and solve them earlier in the project. This helped us avoid dragging dead features with us for too long, or continue to work with broken or semi-functional features.
>
> On other projects we didn't have the chance to play test early during production, but we were able to offer closed beta access and gather feedback during this period. Of course, this sometimes meant having to throw away and rework a lot more work than what should've been the case. If we had

caught those issues earlier in production, we could've avoided throwing away bigger parts of the game.

[...] To me, a playtest has the function of seeing how the game is played, how it is perceived, if it does what it is supposed to do, and evokes the emotions that we'd hoped it would. In combination with QA testing [i.e., testing for bugs], it allows us to catch minor and major issues as early as possible, and not waste too much time on fixing these things. The sooner play testing can start, the better.

Kayode Shonibare-Lewis, Developer at KnapNok Games, UK/Denmark:

A playtest is when you get people to play a game that's in development in order to reveal aspects of the game that may be missed when the game is played by the developers. This could include usability issues, inappropriate difficulty levels and bugs.

Playtests are useful to use whenever the game has playable mechanics or features that are functioning to a level where the developer could get useful feedback. In the early stages of development there may be a lot of feedback about missing features or bugs that the developer is already aware of. It can however still be useful to get feedback as early as possible to have enough development time to act on any required changes. Later on in the development process, when more of the features are in place it is important to start more rigorous testing in order to really catch any outlying problems. The players are observed and notes are taken. Design decisions have to be made based on how the players act in comparison to the desired experience.

We are using playtests to get feedback on how people who have never played the game before understand the interface and puzzles. We are looking for major issues, especially ones that multiple testers repeat, and possible solutions. It's hard to say when we typically use playtesting more specifically than when we feel we need external feedback in order to progress with design changes.

We invite people to come into the office to play the game. We then sit them in front of the television and let them play the game as they are observed. They are also told the feel free to say anything that they are thinking while they are playing. After the tests, I ask them a few questions about how they found the experience. Playtests last for about an hour and I usually don't give the players any help unless they ask (and even then I only point them in the right direction). Testers usually don't ask for help and eventually figure things out.

Martin Fasterholdt, Game Designer at Playdead, Denmark:

During production we make lots of assumptions about how various parts of the game will play out. Playtesting is essential for confirming or refuting these

assumptions. Furthermore playtesting highlights unforeseen complications both in terms of bugs, but more importantly in how gameplay flows. Lastly playtesting inspires new ideas, which emerge from players interacting in certain an unexpected ways.

Each session has between two and five participants. They play through the game in its current state. If needed they skip parts not ready to be tested yet, but in ideal sessions they play from start to finish. Data gathering is done by observing how players interact with the game as well as their body language. The QA manager as well as a game designer observes the testers and makes notes.

Playtesting is conducted only occasionally early in the project. As the project progress the frequency is increased. For about the last year and a half of the project, testing takes place every week.

One of the challenges we face with the way we playtest is observing the relevant aspects of any given session. There is so much information and pin pointing the most import parts, its all up to the observing designer and the QA manager. Furthermore it can be difficult to interpret what you observe and maybe most challenging of all to know what to adjust to improve the experience.

In terms of limitation we are often more limited by time than resources. We could benefit from running larger sessions, but that would require more observing designers, which would not make sense for the production in general. We could also test more frequently with the current size, but this could easily result in the designers spending too much time not implementing. In terms of intensity once a week with a fairly small group seems like a good balance at the current time in the project.

If we had complete freedom in regards to choosing testers, we would select a more nuanced variety than currently possible. Often we lack people who have not played limbo as well as people who do not play many games in general.

Given infinite resources as well as the time, we would probably experiment a lot with capturing video or perhaps recording replays. However at the moment even if we did take the time to record in various ways we would not have the time to look through it.

Jordan Lynn, Player Experience Manager at Volition, US, briefly describes playtesting as:

> [...] my definition of playtesting definitely includes the inclusion of both qualitative feedback from surveys, observation, and direct interviews and quantitative feedback from our telemetry tools.

Describing the testing process he further elaborates:

> My approach to playtesting is best described as dirty science: get results that are verifiable and pass basic tests of logic and then ship that information to the team as quickly as possible. I never have cross test validity, since I use new builds for every playtest and that introduces new elements into the test condition. I don't use statistically valid sample sizes (n 30) because it takes far too long to analyze and the common core of usability testing states that once you hit 5 playtesters you'll locate 80% of your critical issues. Mostly, I locate an area of confusion or frustration, analyze available data about potential causes, and kick that data to the team.

Martin Pichlmair, co-founder at Broken Rules, Austria and Assistant Professor at the ITU Copenhagen, Denmark explains their approach to playtesting:

> In some projects we did playtesting from very early on, when we had the first vertical slice of the game, when we had more than just a prototype, but an outlook onto how a final level would look like. In earlier projects we also did playtesting at an earlier stage when we had to decide between multiple prototypes
>
> We usually do it starting with the vertical slice because many people are having a hard time to test a game when it is lacking feedback [to the player's actions] or nice graphics. Most testers then only remark about the missing graphics and that it would be nice if the game looked nicer. That's all things you already know as a developer, but nothing new.
>
> In these early stages watching the testers play is more important than the feedback they articulate after the session. Having user feedback early in a project is very important, but you can get much more out of the testers if the project has progressed a bit further.

Bibliography

Amaya, G., Davis, J. P., Gunn, D. V., Harrison, C., Pagulayan, R. J., Phillips, B., and Wixon, D. (2008). Games User Research (GUR): Our Experience with and Evolution of Four Methods. In Isbister, K. and Schaffer, N., editors, *Game usability advice from the experts for advancing the player experience*, chapter 4, pages 35–64. Morgan Kaufmann.

Bootsma, R., Fernandez, L., and Mottet, D. (2004). Behind Fitts' law: kinematic patterns in goal-directed movements. *International Journal of Human-Computer Studies*, 61(6):811–821.

Bostock, M., Ogievetsky, V., and Heer, J. (2011). D^3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309.

Bourdieu, P. (1977). *Outline of a Theory of Practice*, volume 16. Cambridge university press.

Canossa, A. (2013). Meaning in Gameplay: Filtering Variables, Defining Metrics, Extracting Features and Creating Models for Gameplay Analysis. In Seif El-Nasr, M., Drachen, A., and Canossa, A., editors, *Game Analytics*, pages 255–283. Springer London.

Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*. John Wiley & Sons, Inc.

Crockford, D. (2008). *JavaScript: The Good Parts*. O'Reilly Media, Inc.

Drachen, A. and Canossa, A. (2009). Towards gameplay analysis via gameplay metrics. In *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era on - MindTrek '09*, page 202, New York, New York, USA. ACM Press.

Drachen, A., Nacke, L. E., Yannakakis, G., and Pedersen, A. L. (2010). Correlation between heart rate, electrodermal activity and player experience in first-person shooter games. In *Proceedings of the 5th ACM SIGGRAPH Symposium on Video Games - Sandbox '10*, pages 49–54.

Drachen, A. and Schubert, M. (2013). Spatial Game Analytics. In Seif El-Nasr, M., Drachen, A., and Canossa, A., editors, *Game Analytics*, pages 365–402. Springer London.

Fitts, P. M. (1954). The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology*, 47(6):381–391.

Gamberini, L., Spagnolli, A., Prontu, L., Furlan, S., Martino, F., Solaz, B. R., Alcañiz, M., and Lozano, J. A. (2011). How natural is a natural interface? An evaluation procedure based on action breakdowns. *Personal and Ubiquitous Computing*, pages 1–11.

Giddens, A. (1984). *The Constitution of Society: Outline of the Theory of Structuration*. University of California Press, Berkeley.

Guardini, P. and Maninetti, P. (2013). Better Game Experience Through Game Metrics: A Rally Videogame Case Study. In Seif El-Nasr, M., Drachen, A., and Canossa, A., editors, *Game Analytics*, pages 325–361. Springer London.

Harmonix (2005). Guitar Hero (Playstation 2).

Heckerman, D. (1996). A Tutorial on Learning With Bayesian Networks. *Innovations in Bayesian Networks*, 1995(November):33–82.

Holmgård, C., Liapis, A., Togelius, J., and Yannakakis, G. N. (2014). Generative Agents for Player Decision Modeling in Games. *Foundations of Digital Games*.

Hoonhout, H. C. M. (2008). Let the Game Tester do the Talking: Think Aloud and Interviewing to Learn About the Game Experience. In Isbister, K. and Schaffer, N., editors, *Game Usability: Advice from the Experts for Advancing the Player Experience*, pages 65–77. Morgan Kaufmann Elsevier.

Hullett, K., Nagappan, N., Schuh, E., and Hopson, J. (2011). Data analytics for game development. In *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, page 940, New York, New York, USA. ACM Press.

Hullett, K., Nagappan, N., Schuh, E., and Hopson, J. (2012). Empirical analysis of user data in game software development. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '12*, page 89, New York, New York, USA. ACM Press.

Hunicke, R., Leblanc, M., and Zubek, R. (2004). MDA : A Formal Approach to Game Design and Game Research. *Discovery*, 83(3):04–04.

Isbister, K. and Schaffer, N., editors (2008). *Game Usability: Advice from the Experts for Advancing the Player Experience*. CRC Press.

Juul, J. (2005). *Half-real*. MIT Press.

Kaptelinin, V. (1995). Activity theory: implications for human-computer interaction. pages 103–116.

Kim, J. H., Gunn, D. V., Schuh, E., Phillips, B., Pagulayan, R. J., and Wixon, D. (2008). Tracking real-time user experience (TRUE). In *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*, page 443, New York, New York, USA. ACM Press.

Lee, K., Chu, D., Cuervo, E., Kopf, J., Grizan, S., Wolman, A., and Flinn, J. (2014). Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Cloud Gaming. Technical Report MSR-TR-2014-115.

Leont'ev, A. N. (1974). The Problem of Activity in Psychology. *Journal of Russian and East European Psychology*, 13(2):4–33.

Lynn, J. (2013). Combining Back-End Telemetry Data with Established User Testing Protocols: A Love Story. In Seif El-Nasr, M., Drachen, A., and Canossa, A., editors, *Game Analytics*, pages 497–514. Springer London.

Mahlmann, T., Drachen, A., Togelius, J., Canossa, A., and Yannakakis, G. N. (2010). Predicting player behavior in Tomb Raider: Underworld. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG2010*, pages 178–185.

Martinez, H. P., Bengio, Y., and Yannakakis, G. (2013). Learning deep physiological models of affect. *IEEE Computational Intelligence Magazine*, 8(2):20–33.

McCallum, S. and Mackie, J. (2013). WebTics: A Web Based Telemetry and Metrics System for Small and Medium Games. In Seif El-Nasr, M., Drachen, A., and Canossa, A., editors, *Game Analytics*, pages 169–193. Springer London.

Medler, B., John, M., and Lane, J. (2011). Data cracker. In *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*, page 2365, New York, New York, USA. ACM Press.

Murphy, K. P. (2002). Learning Markov Processes. In Nadel, L., editor, *The Encyclopedia of Cognitive Sciences*. Macmillan.

Nacke, L. (2013). An Introduction to Physiological Player Metrics for Evaluating Games. In Seif El-Nasr, M., Drachen, A., and Canossa, A., editors, *Game Analytics*, pages 585–619. Springer London.

Nardi, B. A. (1995). Studying Context: A Comparison of Activity Theory, Situated Action Models, and Distributed Cognition. In Nardi, B. A., editor, *Context and consciousness: activity theory and human-computer interaction*, pages 35–52. Massachusetts Institute of Technology.

Nicolini, D. (2012). *Practice Theory, Work, and Organization*. An Introduction. Oxford University Press, Oxford.

Reckwitz, A. (2002). Toward a Theory of Social Practices: A Development in Culturalist Theorizing. *European Journal of Social Theory*, 5(2):243–263.

Salen, K. and Zimmerman, E. (2003). *Rules of Play: Game Design Fundamentals*. MIT Press.

Schell, J. (2009). Good Games Are Created Through Playtesting. *The Art of Game Design: A Book of Lenses*, pages 389–396.

Seif El-Nasr, M., Drachen, A., and Canossa, A. E. (2013a). *Game Analytics -Maximizing the Value of Player Data*. Springer London, London.

Seif El-Nasr, M., Gagné, A., Moura, D., and Aghabeigi, B. (2013b). Visual Analytics Tools – A Lens into Player's Temporal Progression and Behavior. In Seif El-Nasr, M., Drachen, A., and Canossa, A., editors, *Game Analytics*, pages 435–470. Springer London.

Smith, A. M., Lewis, C., Hullett, K., Smith, G., and Sullivan, A. (2011). An Inclusive View of Player Modeling. In *FDG '11 Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 301–303.

Sundstedt, V., Bernhard, M., Stavrakis, E., Reinhard, E., and Wimmer, M. (2013). Visual Attention and Gaze Behavior in Games: An Object-Based Approach. In Seif El-Nasr, M., Drachen, A., and Canossa, A., editors, *Game Analytics*, pages 543–583. Springer London.

Swain, C. (2008). Master Metrics: The Science Behind the Art of Game Design. In Isbister, K. and Schaffer, N., editors, *Game Usability: Advancing the Player Experience*, chapter 9, pages 119–140. Morgan Kaufmann.

Wallner, G. and Kriglstein, S. (2013). PLATO : Understanding Gameplay Data Through Visualization.

Wallner, S., Pichlmair, M., Hecher, M., and Wimmer, M. (2015). Modeling Routinization in Games - An Information Theory Approach. In *Proceedings of the Second ACM SIGCHI Annual Symposium on Computer-human Interaction in Play*, page pp, London, UK. ACM.

Wittgenstein, L. (1922). *Tractatus Logico-Philosophicus*.

Yamagata-Lynch, L. C. (2010). Understanding Cultural Historical Activity Theory. In *Activity Systems Analysis Methods*, pages 13–26. Springer US, Boston, MA.

Yannakakis, G. N. and Hallam, J. (2009). Real-time game adaptation for optimizing player satisfaction. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):121–133.

Yannakakis, G. N., Spronck, P., Loiacono, D., and André, E. (2013). Player Modeling. In Lucas, S. M., Mateas, M., Preuss, M., Spronck, P., and Togelius, J., editors, *Artificial and Computational Intelligence in Games*, volume 6 of *Dagstuhl Follow-Ups*, pages 45–59. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany.

Zimmermann, T., Phillips, B., Nachiappan, N., and Chuck, H. (2012). Data-Driven Games User Research. In *CHI Workshop on Game User Research (CHI-GUR 2012)*, pages 1–4.