PROBABILISTIC MODELING FOR GAME CONTENT CREATION AND ADAPTION

MIGUEL GONZÁLEZ-DUQUE

A framework for adapting content to users based on
Gaussian Processes and Deep Generative Models

April 2023
Digital Design Department
IT University of Copenhagen

PROBABILISTIC MODELING FOR GAME CONTENT CREATION AND ADAPTION

MIGUEL GONZÁLEZ-DUQUE

miguelgondu@gmail.com
miguelgondu.com/about

This thesis is available online:
miguelgondu.com/assets/phdthesis.pdf

Creative AI Lab
Digital Design Department
IT University of Copenhagen

Supervisors

Prof. Sebastian Risi
IT University of Copenhagen

Prof. Søren Hauberg
Technical University of Denmark

This dissertation is submitted to the Digital Design Department
in partial fulfillment of the requirements for the
degree of Doctor of Philosophy (PhD) at the
IT University of Copenhagen

Submitted:
4th of April, 2023

Defended:
12th of June, 2023

Copenhagen, Denmark

Para Hernán y Diana
con mucho amor,
y con todo el agradecimiento.

RESUMÉ

Forskning inden for dynamisk justering af sværhedsgrader i spil fokuserer på, hvorledes spilindhold kan tilpasses spillerens niveau med henblik på at bevare vedkommende i en opslugt flow-tilstand. Størstedelen af eksperimenter inden for området opererer ud fra metoder, der tilpasser faktorer, såsom modstander-AI eller tilgængelighed af spilressourcer, for derved at maksimere spillerens engagement eller minimere sandsynligheden for, at spilleren opgiver spillet. Disses metoder vedligeholder imidlertid ikke en model af spilleren og benytter ydermere teknologier, der er yderst specifikke for de spil, hvori eksperimenterne udføres (fx kan de være afhængige af, at modstander-AI'ens søgealgoritme har adgang til forward-modeller). Nuværende metoder tillader desuden ikke at indstille sværhedsgraden, hvis spildesignere skulle ønske indhold af højere eller lavere sværhedsgrad.

Denne afhandling fremsætter og undersøger en metode, baseret på bayesiansk optimering, hvori spilindhold kan tilpasses brugeren og dermed give spildesignere større fleksibilitet til at sigte mod et ønsket færdighedsniveau. Med udgangspunkt i alle mulige design, en målbar værdi, forudgående viden om denne værdi samt en idealværdi gennemsøger vores metode hurtigt alle mulige spil-baner/missioner efter en af den ideelle sværhedsgrad (det vil sige, et design, der har den fastsatte idealværdi). Som en del af processen vedligeholder vores metode en enkel data-baseret model af spilleren, som kan bruges til yderligere beslutningstagning og analyse.

Vi undersøger denne metode i to forskellige omstændigheder: Først i tilpasningen af spilniveauer inden for 'dungeon crawler'-spilgenren til AI-spillere baseret på søgealgoritmer så som Monte Carlo tree search og rolling horizon-evolution, og dernæst i tilpasningen af både sudoku og 'dungeon crawler'-spil til menneskelige spillere. Vores metode formår at tilpasse spilindhold til AI-spillere, så længe disses færdighedsniveau ikke er ekstremt, og rammer et passende sudokuspil efter omkring 7 iterationer.

Frem for at lade vores metode være afhængig af spildesigneres angivelse af et reelt tal for indkodning af spilindholdet (fx antallet af placerede cifre fra start i en sudoku), undersøger vi ydermere hvordan denne indkodning kan automatiseres ved hjælp af deep generative maskinlæringsmodeller. Med andre ord udforsker vi designmuligheder ud fra indlærte mulige spildesign fra variationsautomatindlejringsmodeller (variational autoencoders) trænet i spil som Super Mario Bros og The Legend of Zelda ved hjælp af spilbaner udtrykt i skrifttegn.

Endelig adresserer vi den udfordring, at maskinlæringsmodeller ikke altid genererer spilbart indhold, og vi bidrager med en ny metode til at interpolere, stikprøveudtage og optimere spilbare dele af variationsautomatindlejringsmodellernes mulige spildesign. Dette bidrag, baseret på differential geometri, er inspireret af nylige fremskridt inden for robotteknologi og

proteinmodellering. Vi kombinerer disse idéer om generering af spilbart indhold og indholdsoptimering og fremsætter dermed en indsnævret version af bayesiansk optimering, hvori indholdet af mulige spilbare områder optimeres. Vi mener at se et tydeligt kompromis i denne metode: Ved at indsnævre muligt spilindhold til blot spilbare områder reduceres diversiteten af det genererede indhold samt kvaliteten af optimeringens idealværdier.

Med andre ord undersøger denne afhandling bayesiansk optimering og deep generative maskinlæringsmodellers anvendelsesmuligheder i skabelsen og tilpasningen af spilindhold til spillere. Vi udvikler en metode der hurtigt finder relevante baner i alt fra korpusser over baner til autogenereret muligt spilindhold, og vi demonstrerer ved hjælp af eksperimenter med både menneskelige og AI-spillere at denne metode formår at finde passende spilindhold i løbet af få iterationer. Denne metode er let anvendelig og kan benyttes til at skabe spil der lærer af og tilpasser sig til deres spillere.

ABSTRACT

Dynamic Difficulty Adjustment studies how games can adapt content to their users' skill level, aiming to keep them in *flow*. Most of these methods maximize engagement or minimize churn by adapting factors like the opponent AI or the availability of resources. However, such methods do not maintain a model of the player, and use technologies that are highly specific to the games in which they are tested (e.g. requiring forward models for enemy AIs based on planning agents). Designers may also intend to find content that is more difficult/easier on purpose, and current methods do not allow for such targeting.

This thesis proposes and tests a framework for adapting game content to users based on Bayesian Optimization, giving designers flexibility when choosing which skill level to target. Starting with a design space, a metric to be measured, a prior over this metric, and a target value, our framework quickly searches possible levels/tasks for one with *ideal difficulty* (i.e. close to the specified target). In the process, our framework maintains a simple data-driven model of the player, which could be used for further decision-making and analysis.

We test this framework in two settings: adapting content to planning agents based on search algorithms like Monte Carlo Tree Search and Rolling Horizon Evolution in a dungeon crawler-type game, and adapting both Sudoku puzzles and dungeon crawler levels to players. Our framework successfully adapts content to planning agents as long as their skill level is not extreme, and takes roughly 7 iterations to find an appropriate Sudoku puzzle.

Additionally, instead of relying on designers to specify a real-valued encoding of the content (e.g. the number of pre-filled cells in a Sudoku puzzle), we investigate learning this encoding automatically using Deep Generative Models. In other words, we explore design spaces learned as latent spaces of Variational Autoencoders using tile-based representations of games like *Super Mario Bros* and *The Legend of Zelda*.

Our final contribution is a novel way of interpolating, sampling and optimizing in the playable regions of latent spaces of Variational Autoencoders, and addresses the challenge that generative models are not always guaranteed to decode playable content. This contribution, based on differential geometry, is inspired by recent advancements in domains like robotics and protein modeling. We combine these ideas of safe generation with content optimization and propose a *restricted* version of Bayesian Optimization, which optimizes content inside playable regions. We see a clear trade-off: restricting the latent space to playable regions decreases the diversity of the generated content, as well as the quality of the optimal values in the optimization.

In summary, this thesis studies applications of Bayesian Optimization and Deep Generative Models to the problem of creating and adapting game content to users. We develop a framework that quickly finds relevant levels in settings varying from corpora of levels to the latent spaces of generative models, and we show in experiments involving both human and artificial players that this framework finds appropriate game content in a few iterations. This framework is readily applicable, and could be used to create games that learn and adapt to their players.

ACKNOWLEDGEMENTS

In 2018, I emailed Niels Justesen, a then PhD student at ITU working with Sebastian Risi. In it, I asked a couple of questions about a paper he had just written on building bots for playing StarCraft. After a couple of exchanges and a meeting over Skype (!), I asked if I could come visit ITU and write my master's thesis under his supervision.

At first, ITU said no. The fact that I was a visiting MSc student and not a PhD made the bureaucracy more difficult. However, after more emails and meetings my visit was approved, and I set foot in Copenhagen for the first time in February 2019. The visit was so fruitful that Sebastian invited me to apply for a PhD position. I was accepted in July and came back to Denmark in late September 2019.

For being so kind as to answer a random Colombian mathematician's cold email, I would like to start by thanking Niels. Our collaboration quickly turned into friendship during those 2 months in early 2019, and it set the first steps of my PhD in motion.

Next, I would like to thank my supervisor Sebastian for his support from the very beginning. I leave ITU having learned from one of the best. Sebastian has always been patient, professional and present. Most of all, I thank him for giving me the opportunity to pursue my curiosity with complete freedom.

In 2021, midway through my PhD, I cold-emailed Søren Hauberg about some applications of his work on differential geometry to content generation in games. He replied by inviting me to give a talk to his group, after which I joined his lab as a visiting student. Søren is an incredible researcher, and I thank him for welcoming me into his group, for his supervision, and for his kind words and brilliant insights.

Both Sebastian and Søren strive to make their research groups as welcoming and social as possible. I am really thankful for the friends I made in both: Elías Najarro, Djordje Grbic, Katt Walker, Rasmus Berg Palm, Claire Glanois, Louisa di Felice, Henrique Galvan Debarba, Rodrigo Moreno, Alison Pouplin, Dimitris Kalatzis, Jeppe Theiss Kristensen, Rosemary Lee, Mads Johansen, Joachim Winther Pedersen, Laurits Dixen, Thomas Volden, among many others.

Copenhagen has welcomed me with open arms during my studies, and I consider myself incredibly lucky for the friends I have made during these three-and-a-half years. Starting with Sophie Thorkildsen, Dom Ford and Simon Bøg, with whom I have shared so many wonderful dinners, beers, boardgame nights, and laughs. Another friend I'm happy to have close is Santiago Quintero, who is like a brother to me. I also cannot forget to mention and thank Sergio Garrido for his friendship, and for so many insightful

chats about probabilistic modeling. Plenty more friends fit this list: Joleen Blom, Priscila Santos da Costa & Luca Lemoni, David Wegmann, Petros Ioannidis, Nina Croitoru, Piyakorn Koowattanataworn, and Imke Grabe. I would also like to thank Johanna for all the love, support and silliness in the final months of this dissertation. I'm happy to be surrounded by all of you.

In 2022, I took a leave of absence to work at the *Bosch Center for AI*, close to Stuttgart. I am glad to have met and been supervised by Leonel Rozo, from whom I learned plenty. During my stay in Germany, I was also lucky enough to make a couple of friends among my colleagues, like Noémie Jaquier, Hadi Beik-Mohammadi, David Adrian, Jan Wöhlke, Aleksandar Taranovic and Leo himself.

Close to the end of my PhD I visited Malmö University, and I would like to thank José María Font for welcoming me and inviting me to give a presentation in their Computer Science department. José María gave me thorough feedback on several chapters of this dissertation and helped me guide the writing process. Thanks!

Finally, my family has also been a great source of support before and during my studies. The reason I'm here is thanks to the hard work of both my parents, Hernán and Diana; the reason I became an academic in the first place is because of my older brother Daniel, with whom I keep geeking to this very day.

Contents

List of Figures

List of Tables

ACRONYMS

AI	Artificial Intelligence
ARD	Automatic Relevance Detection
ARM	Autoregressive Model
BO	Bayesian Optimization
CMA-ES	Covariance Matrix Adaptation - Evolutionary Strategy
DDA	Dynamic Difficulty Adjustment
DDGM	Diffusion-based model
DGM	Deep Generative Model
EBM	Energy-based Model
EDPCG	Experience-Driven Procedural Content Generation
EI	Expected Improvement
ELBO	Evidence Lower Bound
ES	Evolutionary Strategy
FBM	Flow-based Model
FR	Fisher-Rao (metric)
GAN	Generative Adversarial Network
GP	Gaussian Process
GPLVM	Gaussian Process - Latent Variable Model
GPR	Gaussian Process Regression
GTS	Greedy Tree-Search
GVGAI	General Video Game AI (framework)
IG	Information Geometry
ITAE	Intelligent Trial-and-Error
KL	Kullback-Leibler (divergence)
LSTM	Long Short-Term Memory
LVE	Latent Variable Evolution
MAML	Model-agnostic Meta Learning
MAP-Elites	Intelligent Trial-and-Error
MCTS	Monte Carlo Tree Search
OLETS	Open Loop Expectimax Tree Search (an agent in the GVGAI Framework)
OSLA	One-Step Look-Ahead
PCA	Principal Component Analysis
PCG	Procedural Content Generation
PCGML	Procedural Content Generation via Machine Learning

RBF	Radial Basis Function (kernel)
RBO	Restricted Bayesian Optimization
RHEA	Rolling Horizon Evolution
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RS	Random Search (an agent in the GVGAI Framework)
SMB	Super Mario Bros
SVM	Support Vector Machine
UCB	Upper-Confidence Bound
VAE	Variational Autoencoder
VGDL	Video Game Description Language
VGLC	Video Game Level Corpus
vMF	von Mises-Fisher (distribution)

Part I

INTRODUCTION

1

INTRODUCTION

In June of 2016 Hello Games released the video game *No Man's Sky*, in which players get to explore a universe with 2^{64} planets (Higgins, 2014). According to the developers, every planet is different, with unique vegetation, creatures, and geological formations (Hello Games, 2016). All of this content is generated on-the-fly, using algorithms to deterministically recreate the state of the game.

The technology behind this game is an example of *Procedural Content Generation* (PCG), i.e. the use of algorithms to generate the content of a video game (Shaker, Togelius, and Nelson, 2016). This technology has been in use since as early as the 1980s, with games like *Rogue* and *Elite*.

PCG has become its own research field, gaining relevance in Machine Learning more broadly, as it is an ideal tool to build artificial agents that are more resilient to changes in their environment, or capable of solving more than a single task (Cobbe et al., 2020; Risi and Togelius, 2020).

PCG algorithms, like the ones behind the games mentioned above, span a *design space* made of all the possible content created by the algorithm. The size of these design spaces begs the question: How can we efficiently explore them? More specifically, how can we find content that is personalized to players' preferences or skills?

This question has been at the center of the research fields of *Experience-Driven PCG* (Yannakakis and Togelius, 2011) and *Dynamic Difficulty Adjustment* (Zohaib, 2018, DDA), and is also the focus of this dissertation. Methods for DDA aim at maximizing engagement by adapting the opponent's AI or modifying the level design according to models of the player. These processes are either highly specific to a certain game type or rely on expensive data gathering *before* the system is deployed.

This thesis proposes a framework for quickly adapting content to users, and gives designers the affordance to target a certain measured value like completion time or win rate. Shown in Fig. 1.1, the proposed framework starts with a telemetric to measure, a content generation algorithm, a target for the telemetric, and optionally a "guess" (or prior) on how the telemetric behaves with respect to the content. Our method uses the guess to propose a piece of content, which is then shown to the user, recording a value for the telemetric and storing it in a *playtrace*. These pairs (content, telemetric) are used to update the prior and repeat the loop.

This framework is based on probabilistic modeling. The main component is *Bayesian Optimization* (BO, Shahriari et al., 2016), a black-box optimization algorithm that is able to find the targeted level with only a few queries from the player. For BO to work, a numerical specification of the game con-

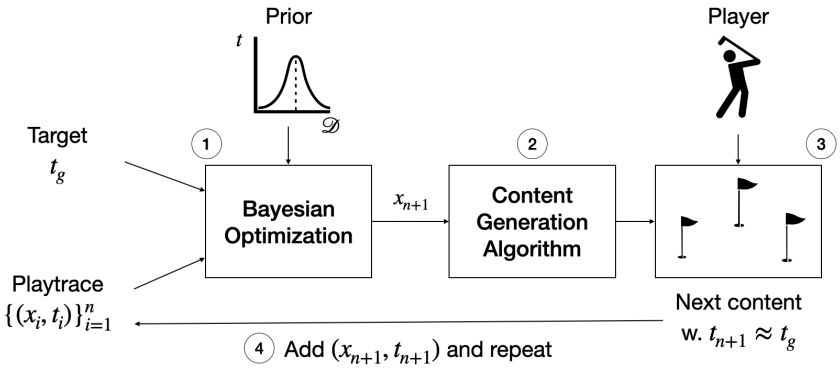


Figure 1.1: **Using Bayesian Optimization to adapt content**. This thesis studies the use of Bayesian Optimization to tailor content to a given player (both artificial and human). Once a target is set and a prior or "guess" is optionally specified, the Bayesian Optimization proposes a certain content specification $\boldsymbol{x}_{n+1}$, the content generation algorithm creates the new content, serves it to the player, and records a metric t_{n+1}. This is stored in a playtrace, which is used to update the prior, repeating the loop. In contrast to prior methods, this framework allows designers to target specific difficulties, does not require access to a game's forward model, and does not rely on gathering playtraces from players before deployment.

tent needs to be provided (e.g. number of enemies and distances to goals in a maze-like game). By using *Variational Autoencoders* (VAEs, Kingma and Welling, 2014), this numerical representation can be learned automatically. VAEs learn to generate novel samples from a given dataset (e.g. Super Mario Bros levels), and in the process learn a continuous low-dimensional *latent* representation of the content. This relaxes the framework's requirement of a content generation algorithm to just a corpus of levels.

Unfortunately, applying VAEs to video game content comes with a challenge: these Deep Generative Models (DGMs), usually applied to images of faces or animals (Karras et al., 2020), learn to generate samples of content that *does not have any functionality requirements*; on the other hand, video game content like game levels need to be functional (i.e. have keys and doors, or be traversable) (Liu et al., 2020; Summerville et al., 2018). **We address this problem** by taking inspiration from methods developed in other fields like robotics and biology, and propose a solution based on differential geometry.

This chapter states the research hypotheses of this thesis, lists its contributions, and covers the layout of the thesis, providing a summary of each chapter.

1.1 RESEARCH HYPOTHESES

The main research hypothesis of this dissertation is the following:

> Bayesian Optimization is a competitive alternative for dynamically (and safely) adapting content to users due to its sample efficiency.

We test this research hypothesis on several settings: first on planning agents and a simple dungeon crawler game in Chap. 4, then on actual players and two games: Sudoku and the same dungeon crawler in Chap. 5.

When improving the framework by adding DGMs to automatically learn the numerical representations of content, the problem of reliably generating functional content arises. We formulate a secondary hypothesis on this front:

> Modifying the geometry of the latent space allows for interpolating, sampling, and optimizing playable content reliably in VAEs trained on tile-based video game levels.

This research hypothesis is explored in Chap. 9, with all the necessary background explained in the chapters leading up to it: Chap. 6 introduces DGMs (and VAEs in particular), and Chap. 7 discusses the mathematical tools for modifying the geometry of the latent space.

1.2 LIST OF CONTRIBUTIONS

This section contains a list of all the research contributions that make this thesis. During these three-and-a-half years, I participated in 9 research projects, all of which lead to publications. This thesis describes 4 of them in depth:

1. „Finding Game Levels with the Right Difficulty in a Few Trials through Intelligent Trial-and-Error," by Miguel González-Duque, Rasmus Berg Palm, David Ha, and my supervisor Sebastian Risi. Presented at the *Conference on Games* (CoG) 2020, and awarded best paper runner-up.
2. „Fast Game Content Adaptation Through Bayesian-based Player Modelling" by Miguel González-Duque, Rasmus Berg Palm, and Sebastian Risi. Presented at CoG 2021.
3. „Pulling back information geometry" by Georgios Arvanitidis, Miguel González-Duque, Alison Pouplin, Dimitris Kalatzis and my co-supervisor Søren Hauberg. Presented at the *International Conference on Artificial Intelligence and Statistics* (AISTATS) in 2022. All authors share first authorship.
4. „Mario Plays on a Manifold: Generating Functional Content in Latent Space through Differential Geometry" by Miguel González-Duque, Rasmus Berg Palm, Søren Hauberg, and Sebastian Risi. Presented at CoG 2022.

Besides these, I also participated in the following research projects during my PhD:

- "Learning a behavioral repertoire from demonstrations" by Niels Justesen, Miguel González-Duque, Daniel Cabarcas Jaramillo, Jean-Baptiste Mouret, Sebastian Risi. Presented at CoG 2020.

- "Towards a framework for human-AI interaction patterns in co-creative GAN applications" by Imke Grabe, Miguel González-Duque, Sebastian Risi, and Jichen Zhu. Presented at the *3rd Workshop on Human-AI Co-Creation with Generative Models* (HAI-GEN 2022) at the *Intelligent User Interfaces* conference.
- "Variational Neural Cellular Automata" by Rasmus Berg Palm, Miguel González-Duque, Shyam Sudhakaran and Sebastian Risi. Presented at the *International Conference on Learning Representations* (ICLR) 2022.
- "Bringing robotics taxonomies to continuous domains via GPLVM on hyperbolic manifolds" by Noémie Jaquier, Leonel Rozo, Miguel González-Duque, Viacheslav Borovitskiy, and Tamim Asfour. This paper is still a preprint.[1]
- "MarioGPT: Open-Ended Text2Level Generation through Large Language Models" by Shyam Sudhakaran, Miguel González-Duque, Claire Glanois, Matthias Freiberger, Elias Najarro, and Sebastian Risi. At the time of writing, this paper is under review.

1.3 THESIS LAYOUT & SUMMARY OF CHAPTERS

This thesis is composed of 4 parts. The first one contains two chapters: this introduction, and related work on content adaption; the second part discusses our first two contributions, in which we test the proposed framework for content adaption on planning agents and human players; the third part focuses on the problem of finding numerical representations of game content automatically, and reliably generating playable content; the final part includes a broader discussion, points to possible future work, and concludes. Fig. 1.2 visualizes these parts, including all the chapters and their interdependencies.

CHAPTER 2 introduces the problem of adapting game content to users, surveying the relevant research on *Experience-Driven PCG* and *Dynamic Difficulty Adjustment*, placing our contributions in context.

CHAPTER 3 provides the mathematical and algorithmic background for the framework. In it, we introduce Gaussian Processes (GPs) and Bayesian Optimization (BO), laying out the notation and algorithms used in the following two chapters. This chapter serves as a brief tutorial on GPs and BO, including a comparison between BO and *Covariance Matrix Adaptation - Evolutionary Strategy* (CMA-ES). The reader who is already familiar with GPs and BO can skim this chapter or skip it completely since the notation used in this dissertation is standard.

1 I worked on this paper while I was on *PhD Sabbatical*, working at the Bosch Center for AI.

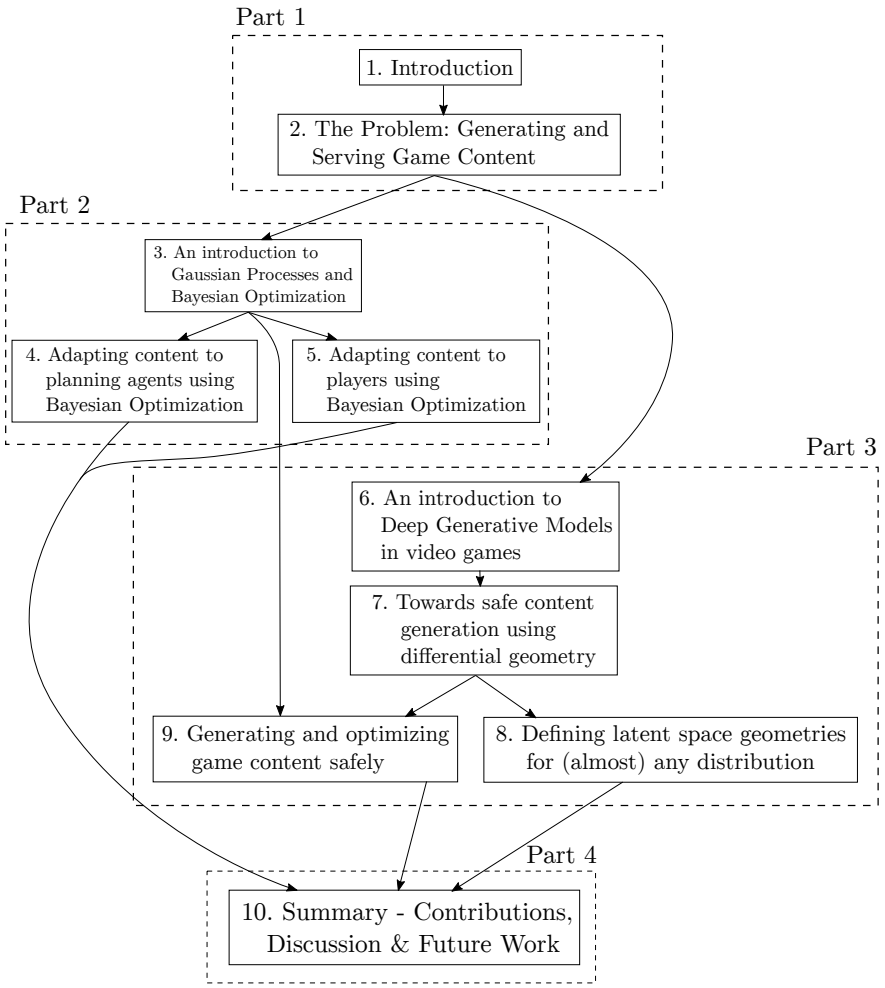


Figure 1.2: **Thesis layout**. This dissertation has 4 different parts. The first one contains an introduction to the problem, the second one deals with applications of Gaussian Processes and Bayesian Optimization, the third one dives into automatic content generation using Deep Generative Models, and how to address the safety problem (i.e. generated content not being playable) using differential geometry, and the last part contains a summary of the contributions and points to future work. The second and third parts can be read almost independently, except for the final experiments in Chap. 9 which leverage Bayesian Optimization. Chap. 8 formalizes the discussion on differential geometry, and provides a new method for defining latent space geometries for (almost) any VAE.

CHAPTER 4 discusses the first application of our framework, which is described in our first contribution (González-Duque et al., 2020). In it, we test using BO for adapting the content of a simple dungeon crawler game to 8 different planning agents inside the General Video Game AI framework (Perez-Liebana et al., 2019b). In this experiment, the prior is made using the *Multidimensional Archive of Phenotypic Elites* (MAP-Elites) illumination algorithm (Mouret and Clune, 2015), and the goal consists of finding a level with approximately 60% win rate for the artificial agents. Results show that, as long as the artificial agents are not over- or underperforming, our framework is able to find a compensatory level in a few trials.

CHAPTER 5 shows an application of our framework to human players. Two games (Sudoku & the simple dungeon crawler from Chap. 4) are used to test our system, measuring its ability to adapt content towards a certain target. In the case of Sudoku, the puzzles are optimized towards puzzles that take three minutes to solve; for the dungeon crawler, the framework optimizes for a target of 10 seconds. A comparison against simpler baselines shows that our framework is a competitive alternative for adapting content to users in Sudoku; the same results are shown for the dungeon crawler, although with more noise and less statistical significance. This chapter covers the methodology and discussion of our second contribution (González-Duque, Palm, and Risi, 2021).

CHAPTER 6 contains an introduction to *Deep Generative Models* (DGMs), the main tool for automatically learning representations of game content. The mathematics of generative modeling are introduced, and the chapter focuses on *Autoregressive Models* (ARMs), *Generative Adversarial Networks* (GANs) and specifically on *Variational Autoencoders* (VAEs), which are the main model used in our experiments. The chapter ends with a discussion on a problem that arises when using DGMs on certain types of game content (like tile-based levels): the generated content needs to be functional, but DGMs learn only the aesthetics. This problem is the focus of the rest of the dissertation.

CHAPTER 7 provides an intuitive introduction to differential geometry, the mathematical toolset used to define safe ways to interpolate, sample, and optimize in latent space. This chapter follows by explaining how the geometry of the latent spaces of VAEs can be modified to make certain regions "expensive" to interpolate through or sample. Applications of differential geometry to other fields like robotics and biology are introduced since they are the inspiration for our methodology. The chapter ends by listing all the challenges of applying these methods in our setting, which are then addressed in the next chapters.

CHAPTER 8 addresses one of the challenges described in Chap. 7, namely that the process for modifying the latent space geometry of VAEs is highly

dependent on the distribution the model decodes to. By using tools from information geometry, the process of defining latent space geometries can be generalized to (almost) any decoded distribution. This chapter covers our third contribution (Arvanitidis, González-Duque, Pouplin, Kalatzis, and Hauberg, 2022) and is fairly theoretical. A reader with no interest in differential geometry or latent space geometries can skip it without interrupting the flow of the dissertation.

CHAPTER 9 discusses how, by modifying the latent space geometry of a one-layer-hierarchical VAE, we are able to interpolate, sample, and optimize playable game content reliably. In this chapter, we train models on levels from *Super Mario Bros* and *The Legend of Zelda*, and test our proposed methods for latent space exploration against simpler baselines. Results show that our modified geometry allows for reliably generating playable content while interpolating, sampling, and optimizing. However, the restrictions we impose on the latent space come with a trade-off: the generated content tends to be less diverse and achieve optima of less quality. This chapter covers an extension to our fourth contribution (González-Duque et al., 2022). We submitted this extended version to the *IEEE Transactions on Games* journal and is currently under review.

CHAPTER 10 summarizes these contributions, provides an overarching discussion, proposes future work, and concludes.

APPENDIX A contains all the technical experimental details (e.g. network specifications, learning rates, and other hyperparameters), which are omitted from the description of the experimental set-ups to ease the reading of the dissertation. These technical details are presented as links to the open-source implementations of the experiments.

2

THE PROBLEM: GENERATING AND SERVING GAME CONTENT

In this chapter, we discuss the related work to our main problem: generating playable game content and adapting it to players. The framework we propose, which is illustrated in Fig. 1.1, is an example of *Experience-Driven Procedural Content Generation* (EDPCG, Yannakakis and Togelius, 2011): the use of algorithms to generate game and adapt game content automatically, personalizing it to a given user's experience.

This personalization of game content can take many forms. Karpinskyj, Zambetta, and Cavedon (2014) separate the work that has been done on this front in five categories: adapting content to users according to their preferences, personality, experience, performance, and in-game behavior. Our method only models the player's performance according to a telemetric specified by the designer and thus can be better described as a form of *Dynamic Difficulty Adjustment* (DDA, Zohaib, 2018).

The goal of this chapter is to put our contributions in context. We start by covering how the framework fits with the current literature on EDPCG and follow by reviewing the state-of-the-art in DDA. The chapter concludes with a summary of this related work, and an outlook for the rest of the dissertation. We postpone the discussion of the related work to deep generative models until *after* we explain the models in Chap. 6, as well as how they fit with our goals of generating *playable* content in Chap. 7.

2.1 EXPERIENCE-DRIVEN PCG

As discussed in the introduction, *Procedural Content Generation* (PCG) is defined as the use of algorithms for the generation of game content (Shaker, Togelius, and Nelson, 2016). PCG systems are varied and have found their way into video games since the early 1980s. Contemporary examples include Spelunky (Mossmouth and BlitWorks, 2020), a game with procedurally generated caves, Minecraft (Mojang Studios, 2011), a sandbox game where players explore, gather resources, and build in massive block-based worlds, Dwarf Fortress (Kitfox Games, 2003) and No Man's Sky (Hello Games, 2016), among others.

Some PCG algorithms are *search-based* (Shaker, Togelius, and Nelson, 2016, Chap. 2): they consider a representation for the game content, which spans a searchable *design space* (or content space). Evolutionary algorithms are often used to explore these spaces (Capps and Schrum, 2021; Earle et al., 2021; Edwards, Jiang, and Togelius, 2021; Giacomello, Lanzi, and Loiacono, 2018; Sarkar and Cooper, 2021b; Sarkar, Yang, and Cooper, 2019; Schrum,

Volz, and Risi, 2020; Shaker, Shaker, and Togelius, 2013; Tanabe et al., 2021; Volz et al., 2018).

Experience-driven PCG searches for content according to a model of player experience. An EDPCG system has the following four components (Yannakakis and Togelius, 2011):

- **A model of the player**, which can be *subjective*, e.g. when users provide their feedback on the experience of playing a game, or *objective*, by measuring in-game behavior or other telemetrics like galvanic response in the skin, which is a proxy for emotional arousal.
- **A measure of content quality**, which uses the player model to assess whether a given piece of content is useful, or fits with the intended goal of the system (for example making a game fun, engaging, or frustrating). This content quality can be evaluated directly from the content using the player model; using agents as proxies for players (i.e. by simulation), or by measuring the interaction of the player with the game.
- **A content representation**, which can be *direct*, e.g. representing platformers' levels using tiles, or *indirect*, where the content is represented by a vector of numbers or a list of positions.
- **A content generator**, which searches the content space to provide the next piece of content which maximizes quality. This maximization process can be exhaustive if the design space is small enough, or can rely on (black-box) optimization algorithms like Bayesian Optimzition (BO), or evolutionary algorithms.

Circling back to our proposed framework (Fig. 1.1), EDPCG gives us the language to describe exactly how our system interacts and adapts to users. Ours is an EDPCG system where the model of the player is objective and data-driven: the value they measure on a telemetric specified by the designer. The content representation is indirect since it is a vector of real numbers from which content can be generated. The content quality is defined by our prediction of the player's performance according to the telemetric,[1] and the content generator is a combination of a designer's provided algorithm for generation and BO.

2.2 DYNAMIC DIFFICULTY ADJUSTMENT

Our framework personalizes game content according to a player's *performance* in the game. Other frameworks that adapt game content in this fashion have been called *Dynamic Difficulty Adjustment* (DDA, Karpinskyj, Zambetta, and Cavedon, 2014; Zohaib, 2018). DDA methods adapt the content of the game to modulate the game's difficulty, aiming to keep players

1 For those already well-versed in BO, the content quality can be seen as the *acquisition function* of the Bayesian Optimization. Chap. 3 gives an introduction to BO and acquisition functions.

in a state of *flow* (where the game is neither too easy to be boring, nor too difficult).

Without using the label "DDA" explicitly, Spronck (2005) considered the online and offline modulation of game AI, adapting to e.g. the player's tactics in RPG games (Spronck, Sprinkhuizen-Kuyper, and Postma, 2004) or in a capture-the-flag task in Quake (Spronck, 2005, Sec. 4.2). For these adaptive systems to be used in practice, Spronck calls for systems that are fast (low computational overhead), effective (need few iterations), robust (to noise inherent in the evaluations) and effective (in achieving what they set out to do, like maximizing engagement).

DDA is established as a research field with two papers by Hunicke, where the system *Hamlet* is proposed and analyzed (Hunicke, 2005; Hunicke and Chapman, 2004). This system uses inventory theory to modulate the access to resources in *Half-Life* (Valve Corporation, 1998), mapping states of the game to actions of the system (e.g. modifying the player's attack strength, or placing items like health packs in the playfield).

Plenty of research has followed since. Zohaib (2018) surveys the field from 2009 to 2018, and finds algorithms for DDA that rely on probabilistic modeling, artificial neural networks, and reinforcement learning, among other techniques. We summarize some of these contributions in the following paragraphs, focusing on the ones most relevant to our approach, and separating them by whether they modulate the opponent's AI or aspects of the level design.

2.2.1 *DDA that modulate AI*

A large body of work has focused on modifying *Monte Carlo Tree Search* (MCTS) bots as a way to achieve adaptivity. Since MCTS algorithms' prediction quality depends on how much compute time it is given, adapting the compute time is a viable way to make the ghosts' AI less powerful in games like Pac-man (Hao et al., 2010; Li et al., 2010; Liu et al., 2009), or a simple chasing game (Sha et al., 2010). Since MCTS is compute-intensive in its predictions, proxy policies trained off-line with MCTS traces have also been considered for this adaptation (Sha et al., 2010).

Instead of modifying MCTS' access to compute resources, (Demediuk et al., 2019) modify the MCTS policy to choose the action with appropriate difficulty, instead of the one that maximizes the bot's chances of winning. This insight is also at the core of our approach since, as we will see in Chap. 4 and 5, we modify an optimization algorithm to target a certain value for a telemetric like completion time, instead of maximizing/minimizing it.

Moon and Seo (2020) train a "meta-learned" model for an opponent AI in an air hockey game using *model-agnostic meta-learning* (MAML). The insight is: meta-learning algorithms can be used to build bots that are *player-agnostic*, i.e. adapt to a player's style easily and with a few gradient steps.

These are types of DDA that focus on modulating the AI inside the game, and thus need access to forward models/offline play traces to train proxy policies. Our framework, on the other hand, modulates levels and not the AIs therein.

2.2.2 *DDA that adapt levels*

An example of a DDA system that works by generating levels is *Polymorph* (Jennings-Teats, Smith, and Wardrip-Fruin, 2010). Based on previous work on generating platformer levels using rhythm (Smith et al., 2009), Polymorph generates chunks of levels continuously and online using statistical models of the chunk's difficulty and of the player's skill. The authors start by collecting several traces with a tool that saves features like whether the level was completed, and how much time the player stood still or walked backward, among others. These were used to train a *Ranked SVM* (a type of regression where the outcome is a ranked list), which is then used to rank possible next levels using the player's current features.

Another example is the use of *Constructive Primitives*, i.e. chunks of platformer levels defined by features like the number and position of gaps, enemies, coins, etc. Shi and Chen, 2017 reduce the dimensionality of these constructive primitives, learn which ones are useful using Active Learning, and choose them dynamically in a DDA set-up. Similarly to us and to (Demediuk et al., 2019), the goal is to minimize a certain "regret", which in this case is the distance to a desired survival rate. The authors formulate this as an optimization problem using a Markov Decision Process, which they solve by using Thompson sampling.

Similarly, Bakkes et al. (2014) also construct a system that adapts platformer levels on-the-fly according to the performance of the player. Their system leverages two policies learned offline: one that maps game states to player experiences defined as a Likert scale from 1 (too easy) to 5 (too difficult) and an exploration policy that returns level parameters informed by the user's annotations. Using these, a policy for online adaption can be deployed: using the exploration policy to construct levels, the goal of this policy is to maximize the probability that the player experience model returns a 3, and minimizes the probability of churn.

All these methods for creating and adapting levels rely on expensive data annotations and offline surveys, explorations, or policy training. Our framework can be deployed without these, as we will see in Chap. 5. We argue that DDA is in a *low data regime* since, as Spronck points out (Spronck, 2005, Sec. 2.3.4), these systems ought to be fast and efficient.

A sample-efficient method for DDA, and perhaps the most similar to our proposed approach, also use BO to quickly model the player and adapt the game content representation to maximize "voluntary time given", or retention. Khajah (2017) deploys two games in Amazon's Mechanical Turk: a Flappy Bird clone, and a "spring ninja" game in which the player must

determine the force to launch a ninja from one platform to another. The authors modify three aspects of e.g. Flappy Bird, two explicit: horizontal and vertical spacing between pipes/platforms, and one implicit: an assistance factor that helps players achieve the goal of the game. This spans a 3-dimensional search space (Khajah et al., 2016).

Using the first traces to build a good general model of retention in this 3D space, the authors deploy BO and find that it is more sample-efficient than random search. Similar content representations and experimental findings are discussed for spring ninja.

Although our framework is also based on Bayesian Optimization, what sets us apart from this work is that, instead of maximizing voluntary time given, our framework allows for targeting specific values of any telemetric provided by the designer. We are bringing the idea of optimizing towards a certain survival rate, win rate, or score for adapting game content using Bayesian Optimization.

2.3 CONCLUSION & OUTLOOK

This chapter introduced the related work on EDPCG and DDA. Using this language, the method proposed in this thesis can be framed as an EDPCG system that targets a certain player's performance. This system uses an objective model of the player (namely, their performance according to metrics like completion time or win rate) which is entirely data-driven. The game content is represented indirectly, using real numbers. The content is optimized towards a certain value for the telemetric, and the content generator leverages BO and either the PCG generator provided by the designer, or a corpus of levels.

Systems that modulate and adapt according to the player's performance can also be framed as methods for DDA, which aim to keep the player in *flow*. In this chapter, we covered related work on DDA systems that modulate either game AI or levels. The key novelty of our approach is the sample efficiency that comes from using BO, and the ability to target specific values of the telemetric (instead of always maximizing).

This chapter postponed the discussion of the related work on deep generative models and the playability issues that arise therein. This related work is discussed thoroughly in Chaps. 6 and 7.

As we discussed in the first chapter, this thesis is structured in 4 parts. The next two parts deal with applications of Gaussian Processes & BO (Chaps. 3, 4, 5), and with applications of deep generative models and differential geometry (Chaps. 6, 7, 8 and 9). These two parts can be read almost independently. The thesis concludes with a final chapter giving a general discussion, and proposing avenues for future work.

Part II

APPLICATIONS OF GAUSSIAN PROCESSES

3

AN INTRODUCTION TO GAUSSIAN PROCESSES AND BAYESIAN OPTIMIZATION

This thesis proposes a framework for adapting content to users based on *Bayesian Optimization* (BO), an algorithm for optimizing black-box functions. In general terms, BO fits a probabilistic model of the *objective function* (the function that is being optimized) and uses it to find the next point to query, balancing exploration and exploitation (Shahriari et al., 2016).

We will focus on a specific type of probabilistic model for our framework: *Gaussian Processes* (GP), a non-parametric regression model known for being sample-efficient and working well in low dimensions, making them an ideal candidate for modeling player behaviors in our framework (Rasmussen and Williams, 2006).

This chapter gives an introduction to GPs and BO, setting up the notation for the rest of the dissertation. Starting with an introduction to GPs, we discuss how they regress functions and predict values using Gaussian distributions defined by a prior and a covariance function, or kernel. These priors, which are initial guesses as to what the modeled function's behavior, can be specified using illumination algorithms like *Multidimensional Archive of Phenotypic Elites* (MAP-Elites) (Mouret and Clune, 2015). Next, BO is introduced, followed with an example: simple optimization testbeds like the `Easom` or `Cross-in-tray` functions (Bingham, 2013). Finally, we give a short comparison of the sample efficiency of BO in these testbeds against a common evolutionary strategy called *Covariance Matrix Adaptation - Evolutionary Stategy* (CMA-ES) (Ha, 2017; Hansen and Ostermeier, 1996).

Since the notation and terminology are standard, a reader with background on GPs and BO could skip to Chap. 4 without losing the thread of the dissertation. The only non-standard part of this presentation is the discussion about building priors using MAP-Elites illumination algorithm (Sec. 3.2), and the comparison against CMA-ES.

The main references for this chapter are *Gaussian processes for machine learning* by Rasmussen and Williams (2006), *Probabilistic Numerics: Computation as Machine Learning* by Hennig, Osborne, and Kersting (2022) and the tutorial „Taking the Human Out of the Loop: A Review of Bayesian Optimization“ by Shahriari et al. (2016). For an interactive presentation of GPs, we recommend the blog posts by Görtler, Kehlbeck, and Deussen (2019) and Deisenroth, Luo, and van der Wilk (2020).

3.1 AN INTRODUCTION TO GAUSSIAN PROCESSES

Given some supervised examples $\mathscr{D} = \{(\boldsymbol{x}_1, f(\boldsymbol{x}_1)), \ldots, (\boldsymbol{x}_N, f(\boldsymbol{x}_N))\}$ (where $\boldsymbol{x} \in \mathbb{R}^d$ and f is real-valued), the goal of Gaussian Process Regression (GPR) is to approximate a function $f(\boldsymbol{x})$ that "captures the pattern" behind the dataset $\mathscr{D}$. Fig. 3.1 shows an example of such a dataset and pattern, which are used as a running example in this chapter: approximating the function $f(x) = x\sin(x)$ given 20 noisy samples.[1]

GPR starts with the assumption that *all finite collections of evaluations of f are normally distributed* with a certain mean and covariance, specified by a *prior function* $\mu_0(\boldsymbol{x}) \in \mathbb{R}$ and a *kernel* (or *covariance function*) $k(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}$. This, we denote $f \sim \text{GP}(\mu_0, k)$ (or just $f \sim \text{GP}$ if the prior and kernel can be inferred from the context).

More precisely, for any dataset of size N,

$$\begin{aligned} f \sim \text{GP}(\mu_0, k) \iff \{f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)\} \sim N(\boldsymbol{\mu}, \boldsymbol{K}) \\ \text{where } \boldsymbol{\mu} = [\mu_0(\boldsymbol{x}_1), \ldots, \mu_0(\boldsymbol{x}_N)], \\ \boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^N. \end{aligned} \tag{3.1}$$

Notice how this imposes a restriction on the covariance function k: it must be *symmetric positive definite* because covariance matrices like $\boldsymbol{K}$ ought to be symmetric positive definite. Symmetric means that $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$, and positive definite means that any matrix $[k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j}^N$ has all positive eigenvalues.

This definition allows us to predict **not only** a point estimate for f at a new point $\boldsymbol{x}_*$, but rather an entire distribution for $f(\boldsymbol{x}_*)$ (including how uncertain we are about the prediction). Under the GP assumption in Eq. (3.1), the collection $\{f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N), f(\boldsymbol{x}_*)\}$ is normally distributed with mean and covariance given by:

$$\widetilde{\boldsymbol{\mu}} = \begin{bmatrix} \boldsymbol{\mu}_{1:N} \\ \mu_0(\boldsymbol{x}_*) \end{bmatrix}_{(N+1)\times 1}, \quad \widetilde{\boldsymbol{K}} = \begin{bmatrix} \boldsymbol{K}_{1:N} & \boldsymbol{k}_* \\ \boldsymbol{k}_*^\top & k(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix}_{(N+1)\times(N+1)}$$

where $\boldsymbol{\mu}_{1:N} = [\mu_0(\boldsymbol{x}_i)]_{i=1}^N$, $\boldsymbol{K}_{1:N} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j}^N$ and $\boldsymbol{k}_* = [k(\boldsymbol{x}_i, \boldsymbol{x}_*)]_{i=1}^N$. Using this joint distribution, the conditional distribution of $f(\boldsymbol{x}_*)$ given all of the previous evaluations $\boldsymbol{f} = [f(\boldsymbol{x}_i)]_{i=1}^N$ is given by:

$$\begin{aligned} p(f(\boldsymbol{x}_*) \,|\, f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)) \sim N(\mu_*, \sigma_*^2) \\ \text{where } \mu_* = \mu_0(\boldsymbol{x}_*) + \boldsymbol{k}_*^\top \boldsymbol{K}_{1:N}^{-1}(\boldsymbol{f} - \boldsymbol{\mu}_{1:N}) \\ \sigma_*^2 = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_*^\top \boldsymbol{K}_{1:N}^{-1} \boldsymbol{k}_*, \end{aligned} \tag{3.2}$$

where we used properties of the Gaussian distribution (Hennig, Osborne, and Kersting, 2022, Chap. 3). Fig. 3.1 shows the posterior mean and standard deviation for all $x \in [-10, 10]$ in the running example. When close to

1 The code used for visualizing this running example can be found in the following URL: https://github.com/miguelgondu/examples_in_thesis.

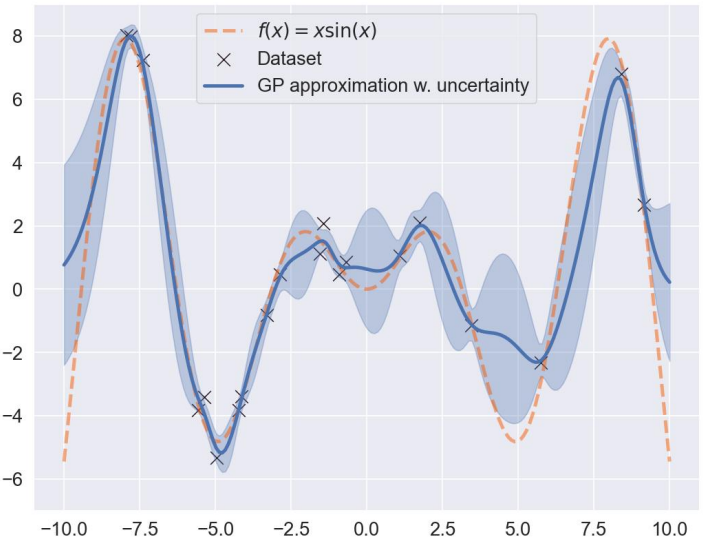


Figure 3.1: **An example of GP regression**. The function $f(x) = x\sin(x)$ (the dashed orange line) is approximated using a GP on a dataset of 20 noisy samples (denoted with X). The GP regression's posterior mean $\mu_*(x)$ is shown in blue, with a shaded region representing the uncertainty of one posterior standard deviation $\sigma_*(x)$ (See Eq. (3.2)), which is higher when there is no data to support the prediction. This access to uncertainty in the predictions is not available in other approximations that are not probabilistic by default, like linear regression or when using neural networks.

previously seen points, the posterior variance σ_*^2 is small and thus the prediction of $f(\boldsymbol{x}_*)$ can be trusted; on the other hand, regions away from the support (like $x = 5$) have high variance.

The kernels used in GPR have associated to them certain hyperparameters which govern global noise or lengthscales that specify how related close-by points are. Noisy observations can be accounted by introducing a hyperparameter σ^2_{noise} in the diagonal of the kernel matrix, replacing $\boldsymbol{K}_{1:N}$ with $(\boldsymbol{K}_{1:N} + \sigma^2_{\text{noise}}\boldsymbol{I}_N)$, where $\boldsymbol{I}_N$ is the identity matrix of size N, in Eq. (3.2).

Fitting a Gaussian Process to a certain dataset means finding the hyperparameters of the kernel that maximize the likelihood of the data. Plenty of software allows for quickly fitting Gaussian Processes on regression, classification and unsupervised learning tasks. In this disseration, we use GPy (GPy, 2012), scikit-learn (Pedregosa et al., 2011) and GPyTorch (Gardner et al., 2018).

GPs are considered *sample-efficient* since they provide good estimates on low amounts of data, and because the complexity of its inference grows with the size of the dataset: computing the inverse of $\boldsymbol{K}_{1:N}$ is $O(N^3)$ in complexity. GPs (as well as other kernel methods) are known to scale poorly with the dimensionality of the input data, and several improvements are being proposed on this front (Eriksson et al., 2019; Gardner et al., 2018; Hensman, Matthews, and Ghahramani, 2015; Maus et al., 2022). As we will see, the usual kernels used for GPR use distance as a proxy for correla-

tion, and the curse of dimensionality implies that distances are no longer as meaningful in high-dimensional spaces (Binois and Wycoff, 2022).

However, our experiments work on the low-data and low-dimensionality regime, and so we rely on the vanilla implementation of GPs by computing the posterior as is specified in Eq. (3.2).

To summarize, GPR works by assuming that finite collections of evaluations of the predicted function f are normally distributed according to a mean and covariance specified by a prior function μ and kernel k. Under this assumption, the function f can be predicted at new values using properties from the multivariate Gaussian distribution. These predictions are not single points, but rather distributions that can be sampled and which have an uncertainty estimate in the form of the variance. Gaussian Processes work well on low-data regimes, and they are known to perform poorly on high dimensional input spaces.

The next two subsections discuss possible choices for priors and kernels, and their impact on the regression.

3.1.1 *Priors*

The prior function $\mu_0 : \mathbb{R}^d \to \mathbb{R}$ in the specification of a Gaussian Process allows us to inject expert knowledge into the regression and to bootstrap our predictions of the function f with an informed guess. When new information arrives, the model changes its predictions if they differ from the prior, and falls back to the prior outside of the *support* of the data.[2]

A common choice for priors in GPs is the zero function $\mu_0(\boldsymbol{x}) \equiv 0$. Indeed, the model itself "picks up" what the shape of the function f should be after updating it with several data points, and every GPR setup can be modified to one that has a zero mean by modeling $f(x) - \mu_0(x)$ instead.[3] Fig. 3.2 shows the running example trained on a restricted version of the dataset (only the points between -2.5 and 2.5 in the domain). Notice that the prediction returns to the prior (which in this case is constant at 0) outside of the evaluations of the function.

3.1.2 *Kernel*

The kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ measures the correlation between two given evaluations $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$ in terms of their inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. These covariance functions are the main design choice when modeling a function with a Gaussian Process since it spans an (infinite-dimensional) family of functions with properties dictated by k (e.g. smooth, periodic, etc.) from which the approximation of f is selected.

2 By support we mean the regions of input space that have data points in them.

3 This follows from the fact that if a vector $\boldsymbol{v} \sim N(\boldsymbol{w}, \Sigma)$, then $\boldsymbol{v} - \boldsymbol{w} \sim N(\boldsymbol{0}, \Sigma)$.

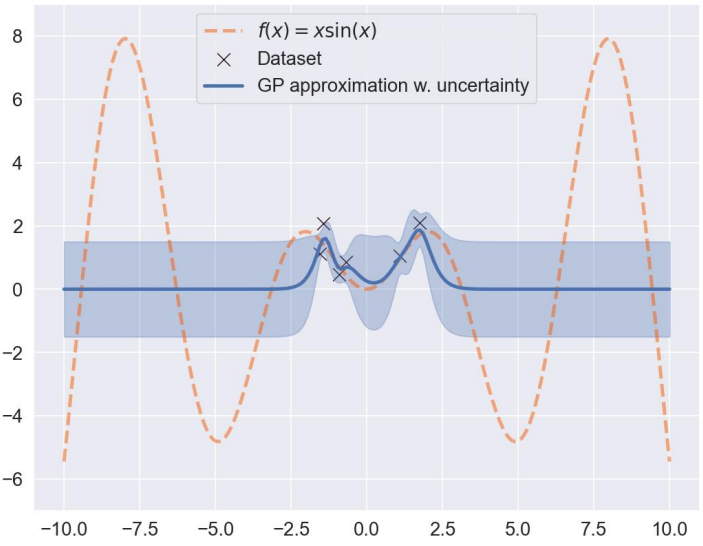


Figure 3.2: **Returning to the prior outside the support**. The prior in a GP allows for specifying domain knowledge, since the prediction will "return to it" outside of the data provided. This figure illustrates this in the running example by restricting the dataset to the $[-2.5, 2.5]$ interval.

In this dissertation, we use the following kernels:[4]

RADIAL BASIS FUNCTION (RBF) KERNEL The RBF kernel uses distance as a proxy for the correlation of the function evaluations. This correlation decays smoothly using a Gaussian bell. Its explicit formula is given by

$$k_{\text{RBF}}(\boldsymbol{x}, \boldsymbol{x}'; \theta_{\text{out}}, \boldsymbol{\theta}_{\text{l}}) = \theta_{\text{out}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^{\top}(\boldsymbol{\theta}_{\text{l}})(\boldsymbol{x} - \boldsymbol{x}')\right), \tag{3.3}$$

where $\theta_{\text{out}} > 0$ is the *output scale*, and $\boldsymbol{\theta}_{\text{l}}$ is a $d \times d$-dimensional matrix with *lengthscales* in its diagonal and zeros outside of it. If all the entries in the diagonal of $\boldsymbol{\theta}_{\text{l}}$ are different, we say that the kernel has *Automatic Relevance Detection* (abbreviated ARD).

Choosing an RBF kernel to approximate a function implies that the function being measured is smooth (i.e. infinitely continuously differentiable).

DOT PRODUCT KERNEL (also known as the Linear kernel) is given by

$$k_{\text{dot}}(\boldsymbol{x}, \boldsymbol{x}'; \theta_{\text{out}}, \sigma_0) = \theta_{\text{out}} \boldsymbol{x}^{\top} \boldsymbol{x}' + \sigma_0^2, \tag{3.4}$$

where σ_0^2 plays the role of a "variance bias", and $\theta_{\text{out}} > 0$ is an output scale.

MATÉRN KERNEL(S) The family of Matérn kernels (Hennig, Osborne, and Kersting, 2022) is given by

$$k_{\nu}(\boldsymbol{x}, \boldsymbol{x}'; \theta_{\text{out}}, \boldsymbol{\theta}_{\text{l}}) = \theta_{\text{out}} \frac{2^{1-\nu}}{\Gamma(\nu)} (\sqrt{2\nu} r)^{\nu} K_{\nu}(\sqrt{2\nu} r), \tag{3.5}$$

4 For a more general introduction, we recommend (Rasmussen and Williams, 2006, Chap. 4), or the blogpost by Duvenand (2014).

where $r = (\boldsymbol{x} - \boldsymbol{x}')^\top(\boldsymbol{\theta}_l)(\boldsymbol{x} - \boldsymbol{x}')$, $\theta_{\text{out}} > 0$ and $\boldsymbol{\theta}_l$ are output and lengthscales (see RBF above), and K_ν is the modified Bessel function of the second kind. In this dissertation we only deal with $\nu = 5/2$:

$$k_{5/2}(\boldsymbol{x}, \boldsymbol{x}'; \theta_{\text{out}}, \boldsymbol{\theta}_l) = \theta_{\text{out}}\left(1 + \sqrt{5}r + \frac{5}{3}r^2\right)\exp\left(-\sqrt{5}r\right). \tag{3.6}$$

Unlike the RBF kernel, using the Matérn family does not imply that f is smooth, thus giving more flexibility to the approximation of the regressed function.

COMBINING KERNELS Covariance functions like the ones described above can be combined to generate new kernels. In particular, given two kernels k_a, k_b and a positive constant $\sigma > 0$, σk_a, $k_a + k_b$ and $k_a k_b$ are also valid kernels (Hennig, Osborne, and Kersting, 2022, pg. 33). Görtler, Kehlbeck, and Deussen (2019) give an interactive presentation of e.g. multiplying kernels, and the impact it has on the regression.

3.2 BUILDING PRIORS USING MAP-ELITES

To mitigate the fact that GPs fail to scale with the dimensionality of the data, Cully et al. (2015) propose learning a GP prior using the MAP-Elites illumination algorithm. Since we leverage this same idea in our first contribution (Chap. 4), this section explains MAP-Elites as a way to construct priors with handcrafted dimensions.

```
procedure MAP-Elites(n_iters, n_init):
    𝒫 = ∅  // a map P(behavior) = performance
    𝒳 = ∅  // a map X(behavior) = genotype
    for iter = 1 → n_iters:
        if iter < n_init:
            x' = random_solution()
        else:
            x = random_selection(𝒳)
            x' = random_mutation(x)
        b' = behavior_descriptor(x')
        p' = performance(x')
        if 𝒫(b') = ∅ or 𝒫(b') < p':
            // update the elite in the cell
            𝒫(b') = p'
            𝒳(b') = x'
    return (𝒫, 𝒳) // behavior-performance map
```

Algorithm 3.1: MAP-Elites' pseudocode.

MAP-Elites explores a high-dimensional space through random mutations, storing the best performing "genotypes" (called *elites*) in pre-defined cells in *behavior space*, which is a hand-crafted description of the genotypes. The result of running MAP-Elites is an archive of elites with diverse behaviors. Since the goal of this algorithm is not to find a single high-performing

individual but rather to understand how elites are distributed in behavior space, MAP-Elites is usually called an *illumination* algorithm.

Suppose the high-dimensional search space is $\mathscr{S}$, then the algorithm requires defining five functions to run.

- `random_solution()`, which returns a random genotype in $\mathscr{S}$.
- `random_selection(`L`)`, which selects an element from a set L at random.
- `random_mutation(`$\boldsymbol{x}$`)`, which randomly mutates a genotype $\boldsymbol{x} \in \mathscr{S}$.
- `behavior_descriptor(`$\boldsymbol{x}$`)`, which returns a low-dimensional handcrafted description $\boldsymbol{b} \in \mathbb{R}^d$ of $\boldsymbol{x}$ (usually after running an expensive experiment, see examples below).
- `performance(`$\boldsymbol{x}$`)`, the function to be maximized, or mapped.

The pseudocode of MAP-Elites can be found in Algorithm 3.1. The code maintains two archives $\mathscr{P}$ and $\mathscr{X}$: $\mathscr{P}$ keeps track of the performance of elites that are close to a certain behavior description $\boldsymbol{b}$, and $\mathscr{X}$ maintains their respective genotypes

For the first `n_init` iterations, the algorithm randomly samples a genotype $\boldsymbol{x}$ from search space and evaluates their behavior description $\boldsymbol{b}$ and performance p. If the genotype is better performing than the current elite in the cell associated with $\boldsymbol{b}$, the cell is updated to maintain $\boldsymbol{x}$ as the new elite with performance p. After the first `n_init` iterations, high-performing genotypes from $\mathscr{X}$ are selected at random and then mutated.

As an example, Cully et al. 2015 consider the search space of policies for robot gaits in a 6-legged robot. The behaviors $\boldsymbol{b} \in \mathbb{R}^6$ are the different percentages of "use" for each leg, and the performance is the velocity of motion. Notice how, to compute the behavior descriptors and the performance, expensive simulation or deployment is required.

3.3 A TUTORIAL ON BAYESIAN OPTIMIZATION

Gaussian Processes can be used to optimize black-box objective functions, i.e. functions that we can only query, and for which we do not have an analytical closed form. Examples of these functions are the energy of a complex physical system in terms of the arrangement of particles, the time a user will spend on a website in terms of the layout and choice of colors, or the validation set accuracy of a machine learning algorithm in terms of its hyperparameters (Shahriari et al., 2016).

The black-box optimization algorithm that we will discuss in this section is called *Bayesian Optimization* (BO), because it uses a Gaussian Process to set a prior of the objective function, and updates it when new information arrives. Since querying the objective function is expensive, it optimizes a different, easy-to-query function instead: the *acquisition function*. The acquisition function uses the GP approximation and its uncertainty estimates to balance exploration and exploitation, proposing a next point to

query that is potentially close to the optima of the objective function. We explain this in detail in the next subsection.

There are other forms of black-box optimization that are more common in the game AI community, like evolutionary or genetic algorithms (Ha, 2017; Salimans et al., 2017); Sec. 3.4 discusses a small comparison between BO and an evolutionary strategy at the end of this chapter.

3.3.1 *Bayesian Optimization step-by-step*

Setting up the notation, BO starts with an objective function $f : \mathscr{S} \to \mathbb{R}$, a GP prior $\text{GP}(\mu, k)$, and a choice of acquisition function $\alpha : \mathscr{S} \to \mathbb{R}$. The function f is expensive to query, and returns the value we want to optimize; the acquisition function is selected from a table (and the ones used in this thesis are explained in the next subsection), and the prior and kernel in the GP are specified using expert knowledge.

With these, BO works iteratively, proposing a *candidate* $\boldsymbol{x}_*$ at each iteration by maximizing the acquisition function α. This process ends after either a certain number of iterations or after an optimum of a certain quality has been reached. [5]

```
procedure Bayesian Optimization(
    f : 𝒮 → ℝ,  // the objective function
    α : 𝒮 → ℝ,  // the acquisition function
    μ,  // the prior of GP
    k  // the kernel of GP
    max_iters=max_iters  // a bound on the iterations
):
    𝒟 = ∅  // the optimization trace
    for max_iters iterations:
        fit f̃ ~ GP(μ, k) with 𝒟 // an approx. of the obj. function
        x* = argmax_{x∈𝒮} α(x; f̃)  // Optimize the acquisition
        evaluate f(x*)
        𝒟 ← (x*, f(x*))
```

Algorithm 3.2: Bayesian Optimization pseudocode.

Algorithm 3.2 shows the pseudocode for Bayesian Optimization. Using a dataset $\mathscr{D}$ (which starts empty), we fit a GP using the specified prior and kernel, arriving at an approximation $\widetilde{f}$ of the objective function. This GP informs the acquisition function α, for which we find the element of the search space that maximizes it, and call it $\boldsymbol{x}_*$. Finally, the objective function is queried at $\boldsymbol{x}_*$, and the pair $(\boldsymbol{x}_*, f(\boldsymbol{x}_*))$ is added to the dataset $\mathscr{D}$ and the loop starts again. This interactive process ends after either a fixed number of iterations have been made, or after $f(\boldsymbol{x}_*)$ is above a certain threshold specified by the user.

5 The stopping criteria for Bayesian Optimization is a highly researched topic. See for example (Makarova et al., 2022).

Sec. 3.4 shows examples of BO on toy optimization functions and compares it with evolutionary strategies. The next subsection explains the acquisition functions used in this dissertation.

3.3.2 *Acquisition Functions*

A key ingredient in the specification of BO is the acquisition function α, which uses the uncertainty estimates of the Gaussian Process to return an informed guess of where the optima of f could be. This section covers the two acquisition functions used in our experiments.[6]

UPPER-CONFIDENCE BOUND (UCB) Given a point $\boldsymbol{x} \in \mathscr{S}$, the UCB is defined by

$$\alpha_{\text{UCB}}(\boldsymbol{x}; \kappa) = \mu(\boldsymbol{x}) + \kappa\sigma(\boldsymbol{x}). \tag{3.7}$$

In other words, the UCB returns a "confident" guess of where the optimum might be: the posterior mean value $\mu(\boldsymbol{x})$ of the approximation of the objective function, plus a weighted uncertainty $\kappa\sigma(\boldsymbol{x})$, where $\sigma(\boldsymbol{x})$ is the posterior standard deviation and $\kappa > 0$ is a hyperparameter which balances exploration (high values) vs. exploitation (low values).

EXPECTED IMPROVEMENT (EI) The GP posterior gives access to uncertainty estimates like expectations w.r.t. $\widetilde{f} \sim \text{GP}(\mu, k)$. For a point $\boldsymbol{x} \in \mathscr{S}$, the EI acquisition function measures how much of an improvement $\widetilde{f}(\boldsymbol{x})$ provides over the current optima in the dataset $f_{\text{best}} = \max_{\boldsymbol{x} \in \mathscr{D}}\{f(\boldsymbol{x})\}$:

$$\alpha_{\text{EI}}(\boldsymbol{x}) = \mathbb{E}_{\widetilde{f}(\boldsymbol{x}) \sim \text{GP}}\left[\max\left(0, \widetilde{f}(\boldsymbol{x}) - f_{\text{best}}\right)\right]. \tag{3.8}$$

This expectation has closed form in terms of the posterior mean and variance of $\widetilde{f}(\boldsymbol{x})$ (Shahriari et al., 2016, Eq. (44)).

3.4 EXAMPLE: COMPARING BAYESIAN OPTIMIZATION WITH EVOLUTIONARY ALGORITHMS

Evolutionary strategies and BO are both black-box optimization techniques, but we argue that they should be used in different regimes. State-of-the-art algorithms for Bayesian Optimization like *Trust Region Bayesian Optimization* (TuRBO) work on toy problems of at most 200 dimensions (Eriksson et al., 2019), and surveys on the subject state that realistic tests have only been made up to 500 dimensions, with a strong focus on problems below 100 (Binois and Wycoff, 2022, Table 1). On the other hand, evolutionary algorithms like CMA-ES (Hansen and Ostermeier, 1996) have been used to optimize neural networks with more than 64^2 parameters in Reinforcement Learning tasks (Salimans et al., 2017, Sec. 4.1). Evolutionary strategies are

6 This explanation is definitely not exhaustive. See (Shahriari et al., 2016) for a complete tutorial.

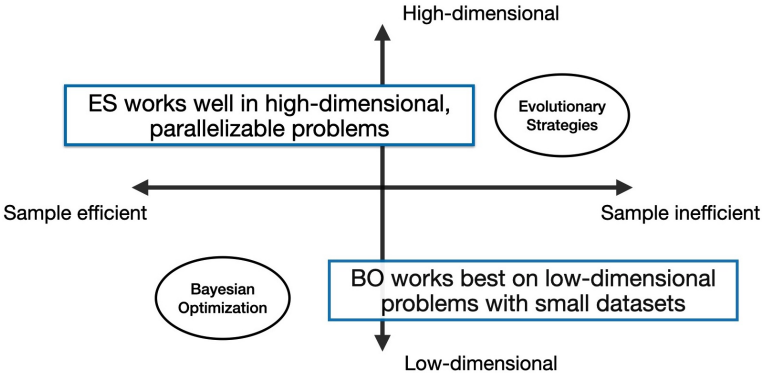
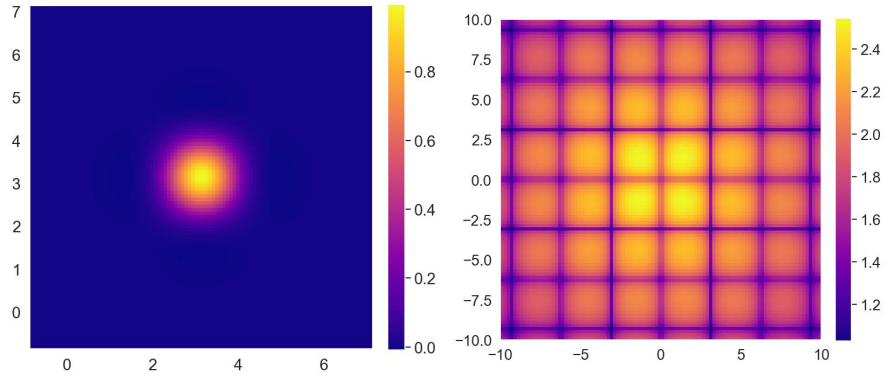


Figure 3.3: **Comparing black-box optimization algorithms.** This figure summarizes a comparison between BO and ES along two axes: sample efficiency and dimensionality of the problem. BO is sample efficient and works best on lower dimensions; ES requires large amounts of compute, but is parallelizable and works well in high dimensions.



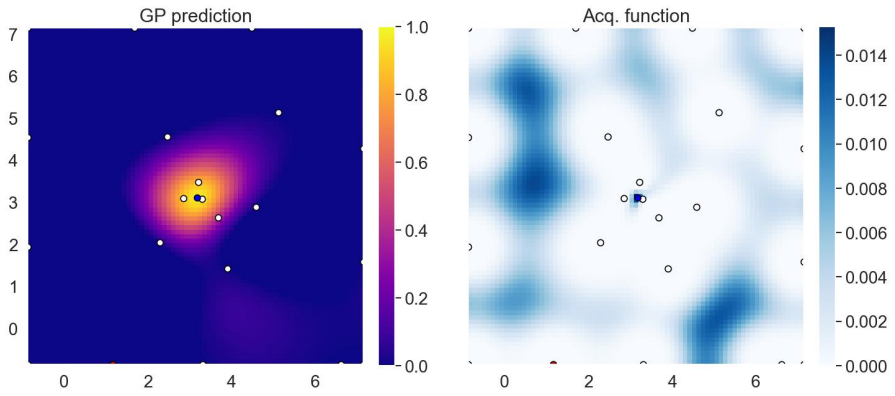
(a) `Easom` (b) `Cross-in-tray`

Figure 3.4: **Two test functions to benchmark black-box optimization algorithms**. This figure shows the `Easom` and `Cross-in-tray` functions (Eqs. (3.9) and (3.10)), which are commonly used as benchmarks of black-box optimization algorithms (Al-Roomi, 2015; Bingham, 2013).

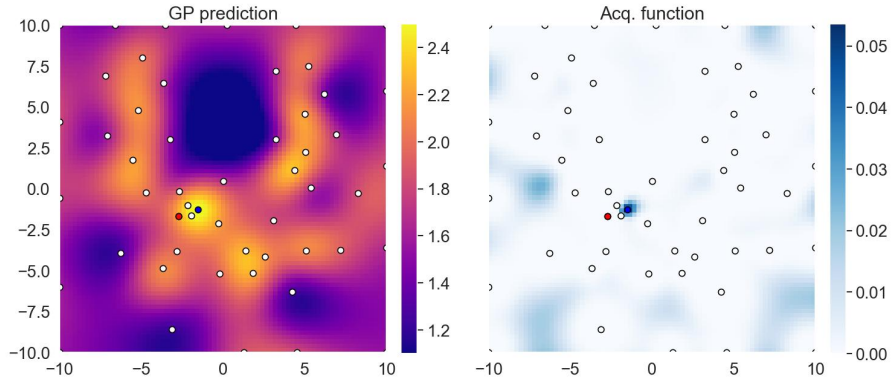
also highly parallelizable, and the process of parallelizing BO depends on the choice of acquisition function (Eriksson et al., 2019). However, evolutionary algorithms are sample inefficient when compared to Bayesian Optimization. Fig. 3.3 summarizes this comparison along these two axes: sample efficiency and tolerance to high dimensions.

To provide examples, this section does a brief comparison between CMA-ES and BO, highlighting the issues raised above.[7] Black-box optimization algorithms are usually tested on a suite of benchmark objective functions which have plenty of local optima, or shapes not amenable to gradient-based optimization (Al-Roomi, 2015; Bingham, 2013). This example com-

7 An implementation can be found here: `https://github.com/real-itu/benchmarking_evolution_and_bo/tree/dissertation-plots/experiments/simple_comparison`



(a) Iteration #23 of BO on `Easom`



(b) Iteration #53 of BO on `Cross-in-tray`

Figure 3.5: **Optimizing benchmark functions using BO**. This figure shows the GP approximation and the EI acquisition function at the last step of a BO run for both `Easom` and `Cross-in-tray`. After 23 iterations, BO has a reasonable approximation of the `Easom` function, and achieves an optimum of 0.998; on the other hand, it takes 53 iterations for BO to find an optimum of 2.540 on `Cross-in-tray`.

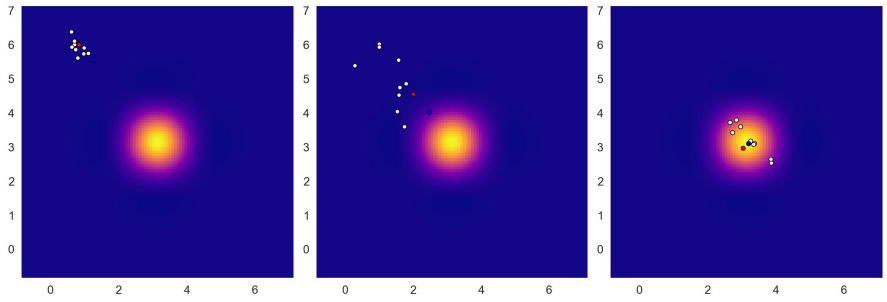
parison focuses on two such functions: `Easom` and `Cross-in-tray`, shown in Fig. 3.4. These functions are given by

$$\texttt{Easom}(x, y) = \cos(x)\cos(y)\exp\left(-(x-\pi)^2 - (y-\pi)^2\right), \tag{3.9}$$

$$\texttt{Cross-in-tray}(x, y) = \left|\sin(x)\sin(y)\exp\left(\left|10 - \frac{\sqrt{x^2+y^2}}{\pi}\right|\right)\right|^{0.1}. \tag{3.10}$$

Fig. 3.5 shows the last iteration for BO when deployed on these two test functions. In this experiment, the optimization stops as soon as an optimum that is at most at 10^{-2} distance of the global optima is reached. Both optimizations start with a flat prior (constant at zero), and by maximizing the acquisition function at each step, they explore the space until finding a suitable optimum.

Compare this with how an evolutionary strategy approaches optimization: by maintaining a mean at the best performing element of the sample space, updating it after sampling from a Gaussian distribution centered at this optimum, and querying the objective function on all the samples (Ha, 2017).



(a) Generation # 2 (b) Generation # 6 (c) Generation #13

Figure 3.6: **Optimizing `Easom` using CMA-ES**. Using CMA-ES with a population size of 10 and a random starting point, it takes 13 generations to find a suitable optimum (i.e. one that is at most 10^{-2} from the global optima). This implies that the objective function was called 130 times. This figure shows the populations in generations 2, 6, and 13.

Fig. 3.6 shows the optimization of the same function using CMA-ES, sampling a population of 10 per generation, or evolution step. This evolutionary algorithm is able to optimize the toy function in 13 optimization steps, using 130 queries to the objective function in total.

We compare the sample efficiency of BO and CMA-ES on 50 different runs of the optimization process on each of the benchmark test functions. The number of points queried on the objective function is measured, and the stopping criteria for both is finding an optimum that is at distance less than 10^{-2} of the global optima. CMA-ES has a budget of 100 generations, and 100 iterations for BO. The `Cross-in-tray` function is difficult for CMA-ES, since the optimization has trouble "escaping" the local optima in each square. Of the 50 runs for `Easom`, CMA-ES found an optimum 41/50 times, and BO 33/50. For `Cross-in-tray`, CMA-ES only found a suitable optimum on 5/50 iterations, and BO always found an optimum. The number of queries BO needs to optimize these benchmark functions was, on average, below that of CMA-ES (Fig. 3.7). In other words, BO is highly sample efficient when it comes to the number of points that the objective function is evaluated on.

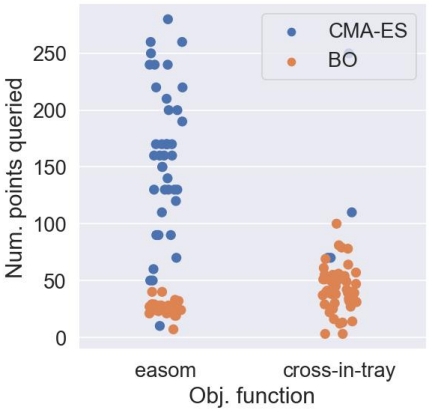


Figure 3.7: **Num. of objective function queries** in CMA-ES and BO. See the main text for analysis.

3.5 SUMMARY & OUTLOOK

This chapter introduces Gaussian Processes (GP) and Bayesian Optimization (BO), the main tools for content adaption in our framework. GPs are an uncertainty-aware method for regression, which starts from the assumption that evaluations of the function are distributed Normally with mean and covariance specified by prior and kernel functions. Using these uncertainty estimates, BO approximates the objective function and iteratively proposes a potential optimum. Compared with evolutionary strategies, BO is sample-efficient; however, literature reports it does not scale gracefully to high dimensions.

The next chapters in this part describe two methods that apply BO to adapt content to users. Chap. 4 uses a prior constructed using MAP-Elites (as described in Sec. 3.2) to adapt content to planning agents, and in Chap. 5 handcrafted priors are used to optimize game content for players.

4

ADAPTING CONTENT TO PLANNING AGENTS USING BAYESIAN OPTIMIZATION

This chapter describes our first contribution (González-Duque et al., 2020), in which we apply Bayesian Optimization (BO) to the problem of adapting game content to planning agents. Due to its sample efficiency, BO is a great candidate for optimizing game content to players: it only needs a few iterations to build a model of the agent, and it can be bootstrapped using expert knowledge.

For a tutorial on Bayesian Optimization see Sec. 3.3.

This experiment starts by building a prior over a corpus of levels evolved using the MAP-Elites algorithm for one agent, which we then use as a prior for BO on *another* agent, aiming to find a level with a certain difficulty. This serves both as an initial test for our core content adaption algorithm as well as a study of different planning agents and their "skill landscape": how well they perform with respect to different aspects of the levels (e.g. amount of enemies, or distance to the goal).

We start the chapter by discussing our motivation: the *intelligent trial-and-error algorithm* (ITAE), first developed in the field of robotics for adapting robot movement to changes in the morphology or environment (Cully et al., 2015). We then dive deeper into the description of the method and the experimental setup, followed by results, discussion and limitations.[1] The prerequisites of this chapter are the introduction on Gaussian Processes (GP), BO and MAP-Elites presented in Chap. 3.

4.1 INTRODUCTION: THE INTELLIGENT TRIAL-AND-ERROR ALGORITHM

We consider our approach to be a translation of the ITAE algorithm from the field of robotics to games. We start, then, by discussing what it is and how it was used in robotics (Cully et al., 2015).

In a few words, the ITAE algorithm is an application of Bayesian Optimization, starting with a prior that was evolved using MAP-Elites. As a short reminder, MAP-Elites maintains a corpus of *elites* in a grid of behavioral characteristics (Mouret and Clune, 2015). One example of such elites are specifications of a quadruped robot's gait, with the behavioral features being how much each leg is used. Another example more relevant to us is video game levels in a dungeon crawler, with "amount of enemies" or "sparsity" as possible behavioral characteristics.

MAP-Elites is explained in detail in Sec. 3.2.

1 The code used for these experiments is available on https://github.com/miguelgondu/finding_game_levels_paper

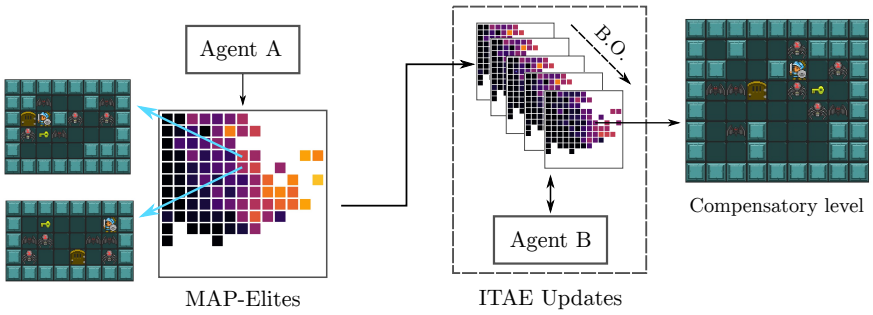


Figure 4.1: **Overview of our first experiment.** First, we evolve a prior using MAP-Elites for a given agent A. Each cell in this prior maintains an elite level with approximately 60% win rate, and we illuminate each cell by how close the elite is to said performance. We show two such elites. In the second phase of our experiment, we use this prior as a proxy for the difficulty of agent B (potentially different from agent A). We use Intelligent Trial-and-Error (ITAE) to query and test level, iteratively updating the prior to adapt to the new agent. After some iterations, we are able to find a level with roughly 60% win rate for agent B.

Once a corpus of elites has been evolved, the ITAE algorithm deploys Bayesian Optimization using it as a starting prior. This bootstraps the optimization to find optima quicker but also allows us to modify the corpus when changes in the environment occur. To test this, the original authors of the algorithm deploy a motion policy on a 6-legged robot, damage the robot, run updates of the corpus using Bayesian Optimization, and manage to find compensatory behaviors quickly.[2]

Our methodology applies this idea to the problem of adapting levels in a dungeon crawler. Instead of evolving robot gaits and changing the environment in which they are deployed, we evolve 2D video game levels and change the planning agent that is solving them (from e.g. an agent that uses greedy tree search to an agent that runs random sampling), with the goal of presenting a level with roughly 60% win rate. Fig. 4.1 shows an overview of this process. The next sections dive deeper into the experimental set-up.

4.2 A BASIC DUNGEON CRAWLER USING GVGAI

The General Video Game AI (GVGAI) Framework allows for quickly compiling and playing games from a text file describing the rules and level (Perez-Liebana et al., 2019a), written in the Video Game Description Language (VGDL, see Schaul, 2013). The developers of the framework ran a competition with several tracks, aimed at testing general video game playing (i.e. agents that can play more than a single game). Researchers have implemented and tested variants of Reinforcement Learning and Planning agents in the framework (Perez-Liebana et al., 2019b, Chapters 4, 5 and 6).

2 For a video covering said paper, check https://youtu.be/T-c17RKh3uE.

To test our content adaption, we used the *Zelda* environment inside the GVGAI Framework.[3] It is a simple dungeon crawler in which the agent must find a key and walk to a door to leave the level. The level can have enemies of different speeds and behaviors.[4] An example of a level for this environment can be found in Fig. 4.2a.

4.3 BUILDING PRIORS FOR PLANNING AGENTS

BO-based algorithms like ITAE can be bootstrapped by using an adequate *prior*, which is a first guess on how the objective function (i.e. the function we are trying to optimize) is specified. As discussed above, the ITAE algorithm evolves this initial form of the objective function using MAP-Elites. In this section we describe this initial step: the simple PCG generator used to create *Zelda* levels, its mutations, behavioral characteristics, and a description of the agents we built priors for.

4.3.1 *A simple PCG generator*

To run MAP-Elites (Algorithm 3.1), two functions need to be specified:

- `random_solution()` which returns, in our example, a random level, and
- `random_mutation(level)` which mutates a level at random.

This section covers their implementation.[5] `random_solution()` starts by sampling width w and height h at random between 3 and 9. 3 is indeed the minimum possible width and height, since levels are surrounded by walls, and 9 was selected after experimentation and visual inspection.

Once w and h have been sampled, there are $f = (w-2)(h-2)$ free positions. Let $i = \min(w, h)$, we sample a random amount of enemies e between $\lfloor i/2 \rfloor$ and i. If $i > 3$, there is enough space to place inner walls, so we sample a number of inner walls w_a between $\lfloor i/2 \rfloor$ and i (otherwise, we set $w_a = 0$).

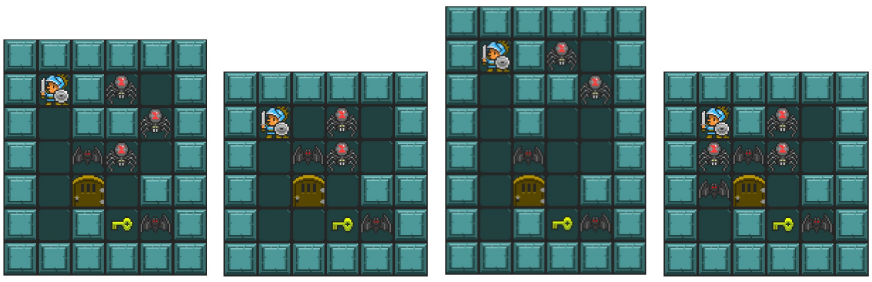
If the amount of free positions f is less than $3 + e + w_a$ (i.e. the avatar, key, goal, enemies and inner walls), we expand the level at random by setting $w = w + 1$ or $h = h + 1$. This ensures that all items can be placed. However, there need to be passable paths between the avatar, key and door.

An empty level of size $w \times h$ is created, placing the edge walls. The avatar, key and goal are positioned at random. We compute the shortest paths from avatar to key, from key to goal, and mark those positions as "occupied" (thus making sure that the level is solvable). All the enemies e are placed at random inside the level, and as many inner walls as possible (up to w_a) are placed at random, without blocking the occupied positions.

3 The VGDL description of this game can be found in (Schaul, 2013, Fig. 2).

4 The actual ontology also includes loot for the player to grab. We simplified the environment by ignoring the loot in our level creation process.

5 These functions were implemented in Python, and the code can be found in the file `zelda_map_elites_utils.py`.



(a) (b) (c) (d)

Figure 4.2: **One random level and its mutations**. In Fig. 4.2a we show the outcome of running `random_solution()` once, followed by the result of randomly mutating the level three times. The mutation procedure removed the third row and two walls in Fig. 4.2b, and a new row and wall were added while removing an enemy in Fig. 4.2c, and two enemies and a wall were added in Fig. 4.2d.

`random_mutation(level)`, on the other hand, randomly expands/shrinks the level in both width and height, as well as randomly adding/removing up to two enemies and inner walls. An example of a randomly generated level can be found in Fig. 4.2a, and three examples of potential mutations can be found in Figs. 4.2b, 4.2c and 4.2d.

4.3.2 *Behavioral characteristics and performance*

We also need to define `behavior_descriptor()` and `performance(level)` to run MAP-Elites. We consider the following behavior descriptors:

- Space coverage: if m is the total amount of tiles inside a level, the space coverage is f/m, where f is the number of non-empty tiles.
- Leniency: how many enemy tiles there are.
- Reachability: the sum of the A star path lengths from the avatar to the key, and from the key to the goal.

These behavioral descriptors are chosen under the hypothesis that they correlate with the difficulty or win rate of the agents. Indeed, the experiments we present in the next sections show that e.g. there is a correlation between win rate and space coverage for agents that only traverse a few steps down the game tree, or that the random agent is only able to reliably finish dense, lenient levels.

On the other hand, and to simulate difficulty adjustment, we decided on a `performance` function that measures the distance to 60% win rate over 40 attempts. We choose a custom function of win rate w that is maximized at this value, given by

$$p(w) = \begin{cases} \frac{5}{3}w & \text{if } 0 \leq w \leq 0.6 \\ -(25/4)\,w^2 + (15/2)\,w - 5/4 & \text{if } 0.6 < w \leq 1 \end{cases} \tag{4.1}$$

This function is visualized in Fig. 4.3. In summary, to compute the `performance` of a given level for a given agent, we let the agent play the level for 40 roll-

outs, arriving at a certain win rate w (i.e. the percentage of winned iterations).

There are several ad-hoc choices in this performance function. The win rate of 60% was selected with the intention of making the levels *slightly easier* since it is well known that GVGAI is a hard challenge and that some of the planning agents struggle to solve levels of even simple games like the one we use. The shape of the function p was also built ad-hoc, with the intention of having a linear increment for levels that were harder than the desired win rate, followed by a quick quadratic drop-off for easier levels.

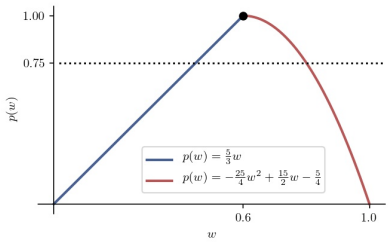


Figure 4.3: **Performance function** $p(w)$.

4.3.3 *Planning agents*

The framework contains several examples of artificial agents, and competitions like the planning track of GVGAI have driven researchers to implement novel variants of planning algorithms (Perez-Liebana et al., 2019a). This section discusses the agents used in our experiment, and provides references to their original implementations. It must be noted that implementing these was not part of our contribution, we rather used already available implementations.

These are split into baseline, basic and advanced agents, according to their skill level. All these agents have access to a forward model, which allows them to simulate (stochastic) steps towards the future (Perez-Liebana et al., 2019b, Chap. 3).

The two baseline agents are simple: `DoNothing`, which does not perform any actions while in the game, and `Random`, which randomly samples actions at each state. The next two sections briefly describe the basic and advanced agents. More details can be found in the implementation itself[6], or on (Perez-Liebana et al., 2019b, Chap. 3 and 4).

4.3.3.1 *Basic agents*

ONE-STEP LOOK-AHEAD (OSLA) is an agent that chooses the next action to take by evaluating all available actions in the current state, measuring the next state with an epsilon-greedy heuristic. The heuristic used takes into account the number of enemies and the distance to the different goals inside the level.

GREEDY TREE-SEARCH (GTS) uses the competition's allotted time to greedily explore the game tree: starting with the current state as the root

6 https://github.com/GAIGResearch/GVGAI/tree/master/src/tracks/singlePlayer.

of the tree, the agent samples actions, simulates them, and repeats the process for the best-performing action. The heuristic this agent uses is the internal state score.

4.3.3.2 *Advanced agents*

RANDOM SEARCH (RS) creates a population of random playtraces (with as many individuals as the allotted competition time allows), and sorts it according to GVGAI's "Win-Score" heuristic, which returns either the current game score or a large positive/negative number if the agent wins/loses.

ROLLING HORIZON EVOLUTION (RHEA) (Perez et al., 2013) evolves a population of sequences of actions. Each sequence stores as many actions as the allotted time allows, and the entire population is evolved using usual mutations and crossover operations over sequences. The same "Win-Score" heuristic is used to assess the fitness of the sequences.

MONTE CARLO TREE SEARCH (MCTS) (Browne et al., 2012) explores the game tree and estimates the values of the different nodes by running simulations until an end state, and propagating the results back. The heuristic stored is the same as in the previous two algorithms.

OPEN LOOP EXPECTIMAX TREE SEARCH (OLETS) (the winner of one of the legs of the GVGAI framework competition in 2014 and 2018 (Perez-Liebana et al., 2019a)) Unlike MCTS, OLETS does not perform rollouts, and it relies on a different heuristic which includes the maximum value of the children of a given node.

4.3.4 *Evolving priors*

Using the MAP-Elites set-up described above, we evolved a corpus of levels for each one of the aforementioned agents. We ran 10 generations per agent, with an initialization of 100 levels, and 50 iterations per generation after that. Since the environments are stochastic, we evaluate the win rate of each level using 40 rollouts. The optimization is guided towards maximizing $p(w)$ in Eq. (4.1).

Fig. 4.4 shows the resulting maps for the different agents, illuminated by win rate. Since we used three behavioral characteristics, we visualize these results in the plane spanned by two characteristics, averaging the remaining one.

These maps summarize the different "skill landscapes" of the agents, e.g. the advanced agents performing well all over, and the basic agents struggling with levels that have too many enemies, or in which the goals are further away. Table 4.1 shows the number of elite levels with win rate w inside different intervals, further supporting this observation.

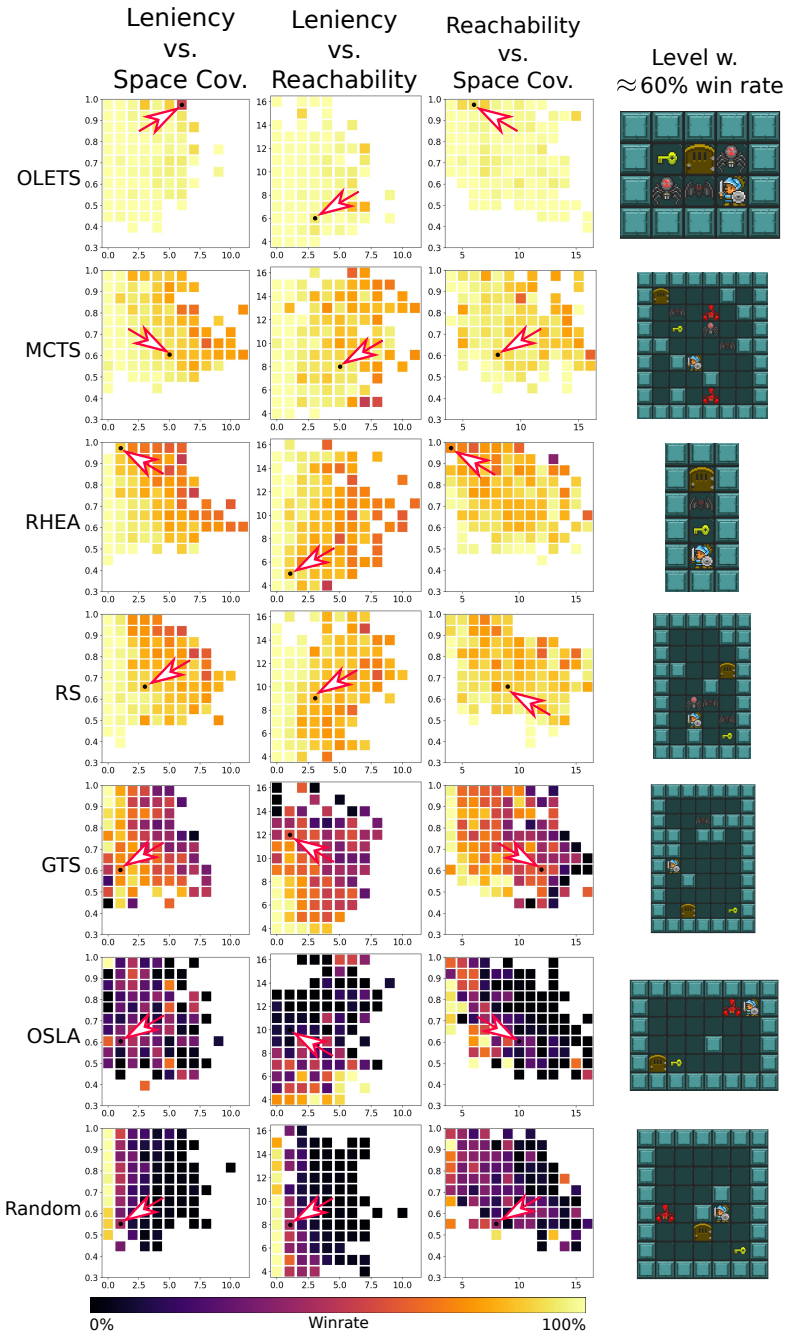


Figure 4.4: **Evolved priors for different planning agents**: This figure shows the final generation of our MAP-Elites procedure designed to evolve levels with a 60% win rate for the planning agents used. The cells are illuminated by win rate, and the red-white arrow points towards a level with roughly 60% win rate, shown to the right. We used three behavioral characteristics, which we will explain using the highlighted elite for Rolling Horizon Evolution (RHEA). *Leniency* counts the number of enemies in the level (1 in this example), *Reachability* adds the lengths of the shortest paths from agent to goals (2 in the agent-to-key path, plus 3 in the key-to-goal path), and *Space Coverage* is the percentage of filled tiles (100%). Each 2D map averages over the remaining feature.

	Easy	Medium			Hard
Agent	$1 \geq w \geq 0.8$	$0.8 > w \geq 0.6$	$0.6 > w \geq 0.4$	$0.4 > w \geq 0.2$	$0.2 > w \geq 0$
OLETS	326	2	1	0	0
MCTS	319	30	5	2	0
RHEA	246	82	13	1	0
RS	268	53	6	1	0
GTS	111	79	69	39	34
OSLA	60	17	22	14	220
Random	48	9	33	41	191
doNothing	0	0	0	0	341

Table 4.1: **Amount of levels per difficulty**: The MAP-Elites procedure evolves levels, aiming at a 60% win rate. This table shows the number of levels segmented by win rate in each evolved corpus. These results show the diversity in skill among the agents, with the advanced ones finding most levels too easy, and the basic agents achieving diversity in their skill landscape. The baseline controllers find most levels too difficult.

4.4 DYNAMIC DIFFICULTY ADJUSTMENT VIA ITAE

As a proxy for testing dynamic difficulty adjustment, we consider the following all-pairs experiment: for each agent, we use the ITAE algorithm to find a "compensatory" level with $p(w) \geq 0.75$ (which corresponds to $w \approx 0.6$, see Eq. (4.1)) in each of the aforementioned priors. Then we test a given agent with the priors of all the other agents.

As a short reminder, the ITAE algorithm is Bayesian Optimization bootstrapped by a prior learned using MAP-Elites. The objective black-box function is the same performance that guided the evolutionary algorithm: each level is tested on 40 rollouts, giving us a certain win rate w which is then passed through the function $p(w)$.

We approximate this objective function using Gaussian Processes with a $\text{Matérn}_{5/2}$ kernel (Eq. (3.6)), implemented using `GPy` (GPy, 2012). The initial kernel hyperparameters are a lengthscale $\boldsymbol{\theta}_l$ with ones in the diagonal, and noise variance given by $\sigma^2_{\text{noise}} = 0.1$.

We run 10 attempts to find a compensatory level using Bayesian Optimization, and we stop after the first 20 iterations. Our BO scheme used the Upper Confidence Bound acquisition function (Eq. (3.7)) with an exploration hyperparameter of $\kappa = 0.03$.[7]

Table 4.2 shows the average amount of BO iterations needed to find a level with win rate w such that $p(w) \geq 0.75$. First, note that the diagonal has low numbers for all prior/agent combinations. This highlights the fact that the elite found for a given agent is indeed a level in which performance is high, meaning there is no need for adaptation. Secondly, the advanced agents (especially `OLETS`) struggle to find a compensatory level in the corpus of basic agents; analogously, `OSLA` and `Random` struggle to find a compensatory level in the priors of more advanced agents. We highlight the

7 Sec. A.2.2 in the appendix contains a summary of these training details.

Prior\Agent	OLETS	MCTS	RHEA	RS	GTS	OSLA	Random
OLETS	1.3 (10/10)	**1.2 (10/10)**	1.4 (10/10)	1.6 (10/10)	7.1 (9/10)	(0/10)	(0/10)
MCTS	11.7 (7/10)	1.2 (10/10)	2.1 (10/10)	2.9 (10/10)	6.7 (10/10)	11.7 (10/10)	(0/10)
RHEA	(0/10)	2.5 (10/10)	1.1 (10/10)	**1.0 (10/10)**	3.2 (10/10)	15.6 (7/10)	(0/10)
RS	**5.4 (8/10)**	2.0 (10/10)	**1.1 (10/10)**	1.5 (10/10)	3.7 (10/10)	12.0 (10/10)	(0/10)
GTS	(0/10)	11.6 (9/10)	7.1 (10/10)	7.5 (8/10)	1.1 (10/10)	11.7 (3/10)	2.6 (10/10)
OSLA	(0/10)	10.5 (10/10)	3.8 (10/10)	5.8 (10/10)	5.8 (10/10)	1.2 (10/10)	20.0 (2/10)
Random	(0/10)	(0/10)	(0/10)	(0/10)	13.7 (3/10)	**3.5 (10/10)**	1.3 (10/10)
doNothing	(0/10)	8.8 (6/10)	3.0 (1/10)	12.0 (3/10)	**2.4 (10/10)**	9.5 (2/10)	11.4 (5/10)
Noise baseline	(0/10)	15.7 (7/10)	3.9 (10/10)	4.5 (10/10)	**1.0 (10/10)**	(0/10)	**2.0 (10/10)**

Table 4.2: **Mean iterations to find a level with ideal difficulty.** This table presents the average number of updates required to find a level with $p(w) \geq 0.75$ for all pairs of priors and agents. We repeat each experiment 10 times, and we present the number of iterations in which a compensatory level was successfully found in less than 20 iterations (e.g. 7/10 means the search found a level with high enough performance in 7 out of 10 runs), together with the average number of updates for the successful iterations. These results show that Bayesian Optimization has potential for content adaption since we are able to find a suitable level in a few iterations. However, whether we are able to find a compensatory level depends on the skill of the agent, with advanced bots like OLETS performing *too well* in all the levels of the basic agent's priors. The same can be said for the Random agent, for which IT&E fails to compensate in the priors of the advanced agents.

lowest non-diagonal term for each agent, noting that a compensatory level can be found usually in less than 15 iterations for most agents (except in the aforementioned cases).

Finally, we discuss the noise baseline. It consists of overwriting the `doNothing` prior with a random performance per cell, effectively shuffling the levels at random. In this random arrangement of levels, the `GTS` and `Random` agents find a compensatory level quickly, while the other agents need roughly more than four iterations.

4.4.1 *Example: a successful attempt*

To illustrate how ITAE quickly finds a compensatory level, let's focus on one of the playtraces that adapt the `RHEA` prior to the `MCTS` agent. Fig. 4.5 shows the three iterations required to find a compensatory level. At first, the system queries two levels that are too easy for `MCTS`; ITAE then updates the prior and queries a level with approximately 0.6 win rate in the 3rd iteration. The queries start with dense, easy levels, but adapt to one with a larger distance to the goal and more enemies.

4.4.2 *Example: an unsuccessful attempt*

It may also happen that no level in the prior works for a given agent. In Fig. 4.6 we show the 1st and 20th updates of a failed ITAE run for `OLETS` on an `MCTS` prior. The performance shown gets dimmer and dimmer as the system realizes no candidate satisfies $w \approx 0.6$.

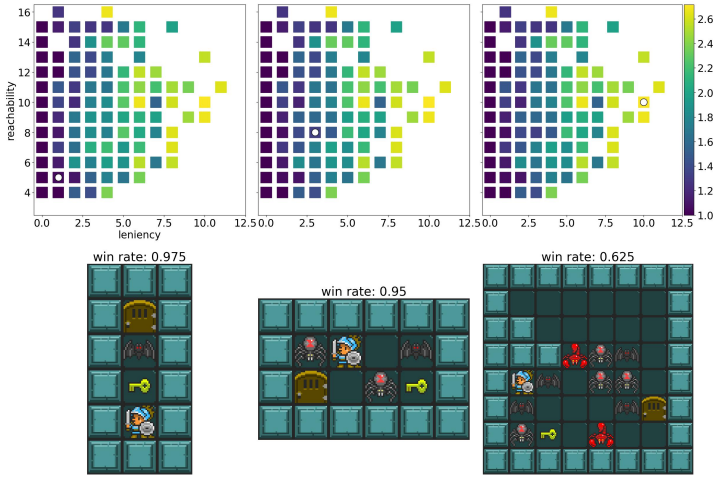


Figure 4.5: **A successful adaptation using ITAE.** This figure shows the RHEA prior, illuminated by performance $p(w)$ (shown in exponential scale to make changes more visible). Starting with this prior, we search for a level that is difficult enough for an MCTS agent. ITAE first queries a dense level with only one enemy (as can be seen in the first column), which MCTS finds too easy. The search then adapts to a harder level shown in the second column, increasing in reachability and leniency. Finally, the search finds a level with approximately 0.6 win rate in the third query.

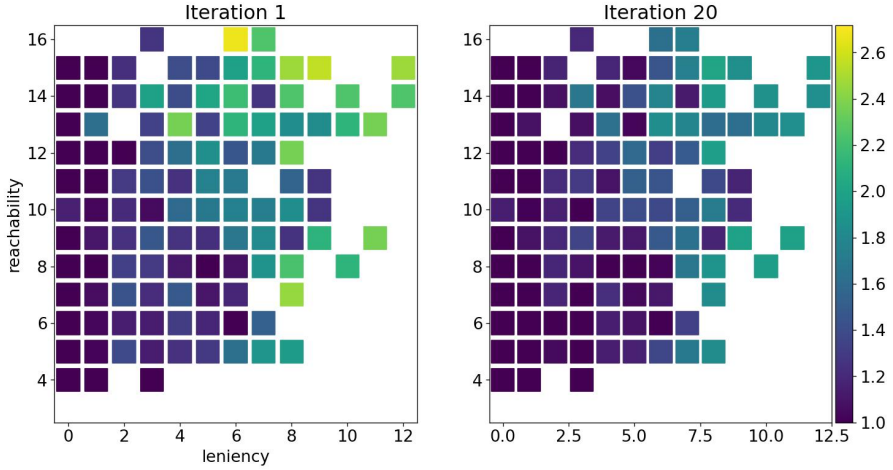


Figure 4.6: **An unsuccessful adaptation using ITAE.** We illuminate the MCTS prior, illuminated by performance $p(w)$ (shown in exponential scale to make changes more visible). In this attempt, no compensatory level was found in the first 20 iterations of ITAE. This figure shows the predicted performance in the first and last iteration, the performance dims as OLETS finds all the queried levels too easy.

4.5 DISCUSSION & LIMITATIONS

This section discusses the results of our experiments. First, MAP-Elites allows researchers and practitioners to gain insights about the "skill landscape" of several planning agents, depending on the behavioral characteristics chosen. In our particular experiment, we choose behavioral characteristics that reflect level density, distance to targets and amount of enemies. With these, we are able to determine that, as one would intuitively expect, shallow-search agents like `OSLA` prefer dense levels in which the targets are close-by (see Fig. 4.4). Similar algorithms could be deployed to understand the current skill landscape of e.g. Reinforcement Learning agents, or players.

Second, we showed an application of the ITAE algorithm to game content adaption. This amounted to applying Bayesian Optimization using an UCB acquisition function and the MAP-Elites priors discussed previously. We show that ITAE is able to find compensatory levels in a simple dungeon crawler for most pairs of planning agents in the GVGAI framework, with exceptions for the agents that are too simple (like the random sampling agent) or well-performing (like the winner of 2014's competition, `OLETS`).

We now summarize some of the limitations of our approach

HANDCRAFTED BEHAVIORAL FEATURES AND CONTINUITY Approximating the performance function p using a Gaussian Process with a Matérn$_{5/2}$ kernel (see Eq. (3.6)) implies an underlying assumption about its continuity. It is plausible that designers choose behavioral features such that the evolved performance is not continuous, making our approach not applicable. In Chap. 9 we explore optimizing content using automatically learned behavioral features using Variational Autoencoders.

SEPARATING THE MODELING FROM THE OPTIMIZATION Our method could be further improved by modeling the win rate instead of the objective function we proposed. To be more precise, we placed a Gaussian Process prior over the performance function $p(w)$, which unfortunately means we can't make any assertions about the win rate w itself. A better idea would then be to predict the win rate of a given level, instead of the transformation $p(w)$.

Separating the modeling of w from the optimization of $p(w)$ would allow us to target *any* win rate. Indeed, we realized that this separation was possible, and we performed subsequent experiments with this in mind (see Chap. 5).

CONTEMPORARY ENVIRONMENT BUILDERS The GVGAI framework seems to be deprecated.[8] Thankfully, there are contemporary alternatives built with Reinforcement Learning in mind. Newer implementations or replica-

8 At time of writing, the GVGAI website `http://www.gvgai.net/` is down.

tions of this experiment could be performed in software like Griddly (Bamford, Huang, and Lucas, 2022) and GriddlyJS (Bamford et al., 2022).

HOW WOULD THIS WORK FOR HUMAN PLAYERS? This experiment serves as a first stepping stone, exploring applications of Bayesian Optimization to adapt content to users. Applying this idea directly would require us to either (1) evolve a MAP-Elites prior using human playtraces, or (2) study how to build agent proxies for human players. The first option would be expensive, but the second one could be explored via imitation learning or Reinforcement Learning (Kristensen, Valdivia, and Burelli, 2020). We explore simpler alternatives that do not rely on MAP-Elites-based priors in subsequent research (see Chap. 5), but future work that directly uses ITAE could rely on a combination of priors, as in (Kaushik, Desreumaux, and Mouret, 2020). In the next chapter, we discuss an application of Bayesian Optimization for adapting content to human players without using MAP-Elites.

5

ADAPTING CONTENT TO PLAYERS USING BAYESIAN OPTIMIZATION

This chapter presents the second application of Bayesian Optimization (BO) to adapting game content, detailing the results of our contribution (González-Duque, Palm, and Risi, 2021).

In the previous chapter, we showed how the Intelligent Trial-and-Error algorithm (which is Bayesian Optimization on top of a MAP-Elites prior) can be used to adapt content between planning agents (Cully et al., 2015; González-Duque et al., 2020). In this contribution, we test our framework in two simple games: Sudoku, and the same dungeon crawler used in the previous chapter. Our system starts by regressing a certain metric (e.g. completion time) using Gaussian Processes (GPs) and then uses this model to choose level specifications that optimize towards a certain completion time using Bayesian Optimization.

This chapter starts by introducing our method with an example in mind, and continues by discussing the technical details that make this system work. Afterwards, the experimental details and data collection are discussed, wrapping up with an analysis and a discussion of the results obtained. Just like in the previous chapter, we assume familiarity with Gaussian Processes and Bayesian Optimization as described in Chap. 3.

5.1 INTRODUCTION: A BAYESIAN OPTIMIZATION FRAMEWORK FOR DYNAMIC DIFFICULTY ADJUSTMENT

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

Figure 5.1: **An example of a Sudoku puzzle**.

To illustrate how our second contribution works, let's consider one of the application examples: Sudoku puzzles. A 9×9 Sudoku puzzle consists of a grid of numbers (some provided, some missing) to be filled out under constraints. No number should appear more than once in its own row, column and 3×3 subgrid. An example of a Sudoku puzzle is shown in Fig. 5.1.

Our system works by "encoding" Sudokus as a certain numerical representation (like the number of missing digits) and searching this design space using Bayesian Optimization. Several numerical encodings could be chosen for Sudokus. For simplicity, we settle on encoding Sudokus by their number of pre-filled digits.

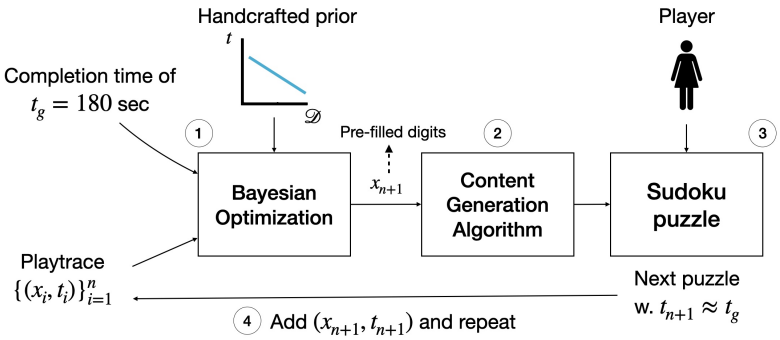


Figure 5.2: **A Bayesian Optimization framework for adapting Sudoku puzzles**. This is an adaptation of Fig. 1.1 to the specific example of Sudokus.

Our Bayesian framework for Dynamic Difficulty Adjustment in this particular example of Sudoku puzzles is shown in Fig. 5.2. The design space $\mathscr{D}$ is the set of all Sudokus in a certain corpus, the specification $x \in \mathbb{R}$ is the number of pre-filled digits, the metric being measured is completion time, the target t_g is 3 minutes (although nothing impedes us from choosing a different goal), and the prior is handcrafted using domain expertise: almost-full Sudokus are easy and take a low completion time, while sparse Sudokus take progressively more time.

After deploying our framework with these settings, our system (1) queries the Bayesian optimization with an empty playtrace, returning an initial number of pre-filled digits x_1 which are optimal according to the prior; (2) generates a Sudoku puzzle with said number of pre-filled digits and (3) presents it to the user, recording how long they took to finish it. This is used to (4) update our playtrace, which is used for the next iteration of the Bayesian Optimization.

Abstracting this process away from Sudoku puzzles, our framework for Dynamic Difficulty Adjustment in games using BO starts with

- a design space (e.g. a corpus of levels or a PCG generator),
- an encoding of this design space, which describes the content using a vector of real numbers $\boldsymbol{x}$,
- a metric $t \in \mathbb{R}$ to be measured,
- a target value $t_g \in \mathbb{R}$ for this metric,
- and a prior over this design space w.r.t. this metric.

With these, the system starts a BO loop that progressively converges toward the specified target value. This framework is visualized in Fig. 1.1, and Fig. 5.2 shows an instance of this framework in the specific case of Sudoku puzzles.

This application of Bayesian Optimization differs from the one we presented in the first contribution (see Chap. 4) on two fronts:

1. The priors were hand-crafted using expert knowledge about the domain, instead of using MAP-Elites.

2. In (González-Duque et al., 2020), our GP surrogate model approximated the "performance function" (a metric of how close the win rate was from 0.6) instead of the win rate itself. Here, we separate the modeling from the optimization and approximate the target metric using a Gaussian Process. This allows us to have a model of e.g. the completion time of Sudokus for a given player, while the previous approach does not provide a model for the win rate of the planning agents we tested.

5.2 MODELLING POSITIVE VALUES

Placing a Gaussian Process (GP) prior over metrics like completion time runs the risk of predicting negative numbers. To force our model to always predict positive values for metrics like completion time, we choose to model $\log(t)$ instead of t. From our model of log-time, we can easily recover the actual times using the usual exponential function for real numbers.

In all the experiments that follow we will still talk about sampling times from the Gaussian Process posterior (denoted $t(\boldsymbol{x}) \sim \text{GP}$), but what we actually mean is sampling $\log(t(\boldsymbol{x})) \sim \text{GP}$ and passing them through the exponential function. We use the former notation for simplicity in the presentation. The choice of kernels and priors will be made explicit during the description of the experimental setup.

5.3 SEPARATING THE MODELING FROM THE OPTIMIZATION

How can we use BO to optimize towards a certain target t_g, instead of maximizing/minimizing? If we were to deploy vanilla BO in the Sudoku example explained above, we would be searching for the Sudoku that *maximizes* the completion time, instead of the one with a completion time closest to the target t_g.

See Sec. 3.3 if you need a refresher on how Bayesian Optimization (BO) works.

To remediate this, we propose a modification of the acquisition function. The setup goes as follows: we are approximating a certain metric $t(\boldsymbol{x})$ using a Gaussian Process $\text{GP}(\mu, k)$, and we want to find values $\boldsymbol{x}$ such that $t(\boldsymbol{x}) \approx t_g$. With this notation consider the Expected Improvement acquisition function α_{EI}, originally given by

$$\alpha_{\text{EI}}(\boldsymbol{x}) = \mathbb{E}_{t(\boldsymbol{x})\sim\text{GP}}\left[\max(0, t(\boldsymbol{x}) - t_{\text{best}})\right],$$

where t_{best} is the highest recorded value of $t(\boldsymbol{x})$ in the playtrace (Eq. (3.8)). Maximizing this would correspond to maximizing the metric $t(\boldsymbol{x})$.

We modify Expected Improvement to allow it to target *any* t_g: define β_{EI} to be

$$\beta_{\text{EI}}^{t_g}(\boldsymbol{x}) = \mathbb{E}_{t(\boldsymbol{x})\sim\text{GP}}\left[\max(0, -(t(\boldsymbol{x}) - t_g)^2 + \tilde{t}_{\text{best}})\right] \tag{5.1}$$

where $\tilde{t}_{\text{best}}$ is the minimum of $(t(\boldsymbol{x}) - t_g)^2$ for the values recorded in the playtrace. Notice how this amounts to evaluating the Expected Improvement

of $\tilde{t}(\boldsymbol{x}) = -(t(\boldsymbol{x}) - t_g)^2$; in other words, this modified acquisition $\beta_{\text{EI}}^{t_g}$ is now maximized at the points $\boldsymbol{x}$ where $t(\boldsymbol{x})$ is closest to t_g.

Recall that the unmodified Expected Improvement α_{EI} has a closed form in terms of the mean and standard deviation of the posterior of the Gaussian Process (see Sec. 3.3.2). This does not happen for our modified version $\beta_{\text{EI}}^{t_g}$, so we have to rely on samples $t(x) \sim \text{GP}$ to compute the expected value in Eq. (5.1).

Something similar can be said about other acquisition functions like the Upper Confidence Bound described in Eq. (3.7). If we modify it to instead maximize $\tilde{t}(\boldsymbol{x})$, we end up with

$$\beta_{\text{UCB}}^{t_g}(\boldsymbol{x}) = -\left((\mu(\boldsymbol{x}) + \kappa\sigma(\boldsymbol{x})) - t_g\right)^2, \tag{5.2}$$

where $\mu(\boldsymbol{x})$ and $\sigma(\boldsymbol{x})$ are the GP posterior mean and variance for $t(\boldsymbol{x})$, and κ is a hyperparameter which governs how much we want to explore unknown regions (where $\sigma(\boldsymbol{x})$ is high): higher values of κ mean that the optimization will explore uncertain regions more often.[1]

5.4 EXPERIMENTAL SET-UP

In this section, we describe the games (Sudoku and Dungeon Crawler) in which the aforementioned framework is tested, how the games are built and exposed to the public, and how the data is collected and processed.

5.4.1 *Description of the corpus, encoding and metrics*

SUDOKU As mentioned above, Sudoku puzzles are arranged in 9×9 grids with 3×3 subgrids, prefilled with a certain amount of digits. The goal of the game is to fill the empty cells with digits between 1 and 9 such that no digit repeats itself in its row, column and 3×3 subgrid. Fig. 5.1 shows an example. We gather all our Sudokus from Kaggle's 9 million Sudoku puzzles and solutions dataset[2] and subsampled only 2000 of them for memory efficiency.

Sudoku is a single-player game with a branching factor that decreases over time: the more cells a player fills, the fewer actions they can take. Moreover, there are no opponents in Sudoku, and the outcome of each move is deterministic.

We encode puzzles by the number of prefilled cells. Sudoku experts (*Good Sudoku*) discuss patterns inside Sudoku puzzles, like naked singles (when a cell only has one possible option) or naked pairs (when two cells in different subgrids share the same options). These patterns could be used to design more elaborate encodings; we settled for the simple encoding given by the number of pre-filled digits. Our framework aimed to serve puzzles

1 Remember that we are fitting $\log(t(\boldsymbol{x}))$ instead of $t(\boldsymbol{x})$, so we transform the sum $\mu(\boldsymbol{x}) + \kappa\sigma(\boldsymbol{x})$ using the exponential.

2 https://www.kaggle.com/datasets/rohanrao/Sudoku.

that took $t_g = 180$ seconds. This target goal was chosen ad-hoc, but the proposed framework would not require any modifications to work with a different target.

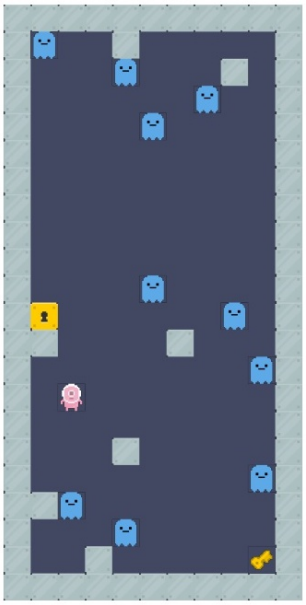


Figure 5.3: **Example of a dungeon level**.

DUNGEON CRAWLER This dungeon crawler we used is a clone of the GVGAI version of *The Legend of Zelda* (see Sec. 4.2). Re-describing it, each level contains an avatar, a key and an end goal. The player must traverse the level, grab the key and reach the end goal while avoiding enemies; the player can kill enemies by swinging a sword in the direction the avatar is pointing. Two differences between the GVGAI version and ours is the way enemies move: enemies move on a per-human-move basis instead of every game loop, and the enemies "face" towards a certain direction, signaling where they might move next. An example of a level is shown in Fig. 5.3, and the exact implementation of this game can be found in the code repositories for this project.[3]

Unlike Sudoku, the branching factor of this Dungeon Crawler game remains constant as the player plays, and there are opponents that move at random. This game, then, poses a greater challenge for our methodology, and it tests how resilient BO is against stochasticity in the evaluations.

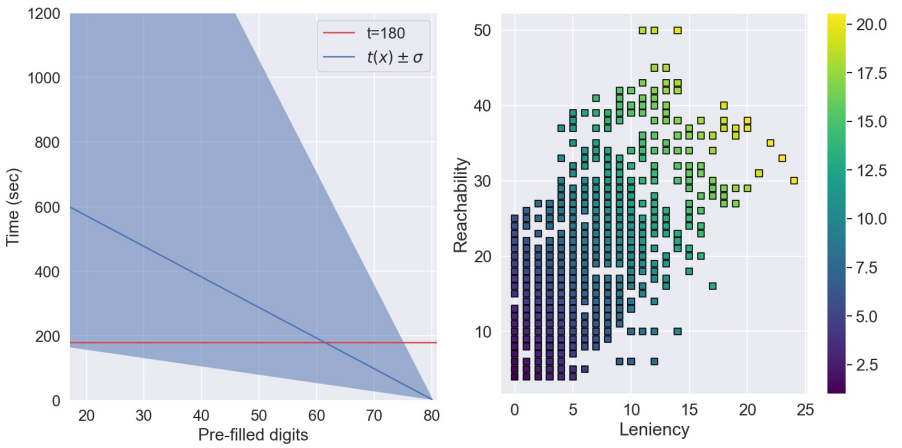
We encoded the levels using only two of the behavioral descriptors originally used for the MAP-Elites evolution in our previous contribution (see Sec. 4.3.2), namely **leniency** (number of enemies in the level) and **reachability** (lengths of the A-star paths between avatar, key and goal). In this experiment, we aimed at presenting levels that took $t_g = 10$ seconds to solve.

5.4.2 *Prior, kernel and acquisition functions*

SUDOKU Our handcrafted prior for the Sudoku task (shown in Fig. 5.4a) was a linear interpolation between the points $(x = 80, t = 3)$ and $(x = 17, t = 600)$.[4] In other words, our prior assumes that a Sudoku with only one missing digit takes only 3 seconds to solve, while a Sudoku with 17 hints (the theoretically minimum number of hints possible) takes 10 minutes to solve. We show this prior in Fig. 5.4a.

3 https://github.com/miguelgondu/bayesian_dungeoncrawler/blob/f9921aa7f94ecf30fcaccc2b5631cea943143596/frontend/src/app/game/game.component.ts#L529

4 $\mu_0(x) = 600 + (x - 17)(600 - 3)/(17 - 80)$.



(a) Sudoku (b) Dungeon Crawler

Figure 5.4: **Priors for both games**. This figure illustrates the prior (i.e. first guess on the player's completion times, crafted using expert knowledge) for both games. The prior for Sudoku states that puzzles that are almost full are easy (passing by $(x = 80, t = 3)$), and puzzles that are sparse are difficult (i.e. $(x = 17, t = 600)$). Similarly, the prior for Dungeon Crawler states that levels without enemies in which the goals are close by are easy to solve, and levels with distant goals and several enemies are difficult. We illuminate the Dungeon Crawler with the completion time specified by the prior.

For the surrogate Gaussian Process model, we used an RBF kernel (see Eq. (3.3)) and modeled $\log(t)$ instead of the completion time itself. With this small modification, the uncertainty of our prior grows as we get further away from the point $(x = 80, t = 3)$, and we are able to always model positive values. We used the modified Expected Improvement $\beta_{\text{EI}}^{t_g}$ described earlier in Eq. (5.1), taking into account that we had to exponentiate before computing the difference with t_g.

DUNGEON CRAWLER To create a prior over the completion time t, we ran the basic PCG generator from our previous contribution and created a corpus of 399 randomly generated levels (see Sec. 4.3 in the previous chapter), with leniencies l ranging between 0 and 24, and reachabilities r bounded from 4 to 50. These levels were assigned a prior value for their completion time by interpolating the points $(l = 0, r = 4, t = 1)$ and $(l = 14, r = 50, t = 20)$[5]; intuitively, this prior says that levels with no enemies and close-by goals can be solved in 1 second, and levels with many enemies and distant goals can be solved in 20 seconds, with the completion time linearly increasing in between. Fig. 5.4b shows this prior, illuminated by completion time.

After several testing iterations, we settled for a combination of the linear and the RBF kernel $k = k_{\text{RBF}} + k_{\text{Dot}}$ (see Sec. 3.1.2), and we used a modified

5 $\mu_0(l, r) = (15/28)l + (1/4)r$ to be exact.

UCB acquisition function as described in Eq. (5.2) with exploration hyperparameter $\kappa = 0.05$.[6]

5.4.3 *Baselines*

SUDOKU We tested our framework against a binary search baseline. Starting with a guess at the middle of the [17,81] interval, this baseline selects the next half of the interval depending on whether the puzzle was too difficult or too easy for a given player. We also gathered traces for a content adaption experiment based on linear regression, the results of which we describe only anecdotally. This linear regression baseline fits a line on the log-times of the users and uses it to select which puzzle to pick in a greedy fashion. The exact implementation of both these baselines can be found in the code repositories for this project.[7]

DUNGEON CRAWLER For the Zelda-like game we implemented two baselines: a noisy hill-climbing algorithm that starts at the center of the prior, takes a step using Gaussian noise and re-centers the distribution if the sampled point has performance closer to the target $t_g = 10$. The second baseline selects levels at random from the corpus. Both implementations are also available in our repository.[8]

EVALUATION METRIC The metric we used to evaluate all these comparisons was the **mean absolute error**, defined as

$$\mu_e(\mathcal{T}; t_g) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} |t - t_g|, \tag{5.3}$$

where $\mathcal{T}$ is a set of times (e.g. the times recorded for our experiment in Sudoku). We abbreviate the absolute error by e.g. μ_e^{Bay} to describe the absolute error of our approach (Bayesian), letting the goal t_g always be 180 for Sudoku and 10 for the Dungeon Crawler. Both these target goals were chosen ad-hoc, but our system allows for modifying them without any change to the methodology.

In our experiments, we will compare the mean absolute error of our approach against that of the baselines, and to do so we establish a null hypothesis H_0^{b} which assumes both means to be equal. We test this hypothesis using a two-tailed t-test and decide to reject it if we achieve a p-value of less than 0.05.

5.5 DEPLOYING THE EXPERIMENT: TWO WEB APPLICATIONS

Both games were built and exposed as web applications. The backend for these was implemented in Python using Flask, and the Gaussian Process

6 Sec. A.2.3 contains a summary of these details.
7 https://github.com/miguelgondu/bayesian_sudoku
8 https://github.com/miguelgondu/bayesian_dungeoncrawler

Iteration	n_{Bay}	n_{bin}	μ_t^{Bay}	μ_t^{bin}	μ_e^{Bay}	μ_e^{bin}	H_0 rejected
1	217	76	142.0 ± 78.0	302.6 ± 272.9	72.5± 47.5	135.9± 266.4	yes ($p = 0.04$)
2	123	41	129.3 ± 110.9	161.2 ± 236.6	76.9± 94.6	122.3± 202.6	no ($p = 0.17$)
3	81	30	161.0 ± 57.1	249.0 ± 379.7	49.6± 33.8	132.7± 361.8	no ($p = 0.22$)
4	49	17	191.0 ± 81.5	160.3 ± 48.4	47.6± 66.7	39.5± 33.2	no ($p = 0.52$)
5	36	13	205.6 ± 127.2	191.9 ± 56.7	60.4± 114.5	44.1± 35.6	no ($p = 0.45$)
6	22	9	168.0 ± 49.6	210.3 ± 127.4	36.5± 34.9	72.6± 106.5	no ($p = 0.35$)
7	16	8	180.8 ± 52.9	186.9 ± 65.6	37.3± 36.3	49.1± 40.1	no ($p = 0.50$)
8	10	6	172.0 ± 42.4	220.7 ± 109.3	29.9± 29.6	79.8± 79.4	no ($p = 0.19$)
All	288	94	153.10 ± 88.8	235.0 ± 255.8	63.9 ± 67.3	110.5 ± 237.1	yes ($p < 0.01$)

Table 5.1: **Average time and absolute errors for Sudoku**. This table shows, for iterations ranging between 1 and 8, the number of unique traces, the average completion time, and the average absolute error for both our Bayesian approach and the binary search baseline.

Regression implementation of `sklearn` (Pedregosa et al., 2011). In both applications, the users were instructed to solve the puzzles/levels as fast as they could.

For the Sudoku experiment we relied on a simple frontend that did not show a timer on the puzzle page, nor did it highlight mistakes made by the users. Once a user submitted a potential solution, our backend analyzed it for its correctness and computed the GP posterior using only the completion times of Sudokus that were solved correctly.

The Dungeon Crawler was cloned for online use using an Angular frontend.[9] As mentioned previously, this game differs from the original inside GVGAI on two fronts: enemies show the direction they are facing, and they only move when the player moves. The movement of the enemies was stochastic and decided at each step of the player; this implies that the same level could have two different difficulties since the enemies would move differently. We familiarized the user with the game mechanics by including three tutorial levels *before* collecting traces, and the GP posterior was also computed by only using the levels that were successfully solved.

These implementations are also open-source and available in our repositories.[10]

5.6 RESULTS

SUDOKU Table 5.1 summarizes the results of our Sudoku content adaption experiment, showing the number of traces, average time μ_t, and average absolute error μ_e for both our approach and the binary baseline.[11] We received 288 unique traces (i.e. sequences of puzzles) for our approach, and 94 for the binary baseline; when considering all these, our average error is significantly lower than that of the baseline.

9 We used and adapted the royalty-free sprites provided by `https://www.kenney.nl/`.

10 See above, or Table A.1 in the appendix.

11 We only considered the first 8 updates and Sudokus that were solved in less than 3000 seconds.

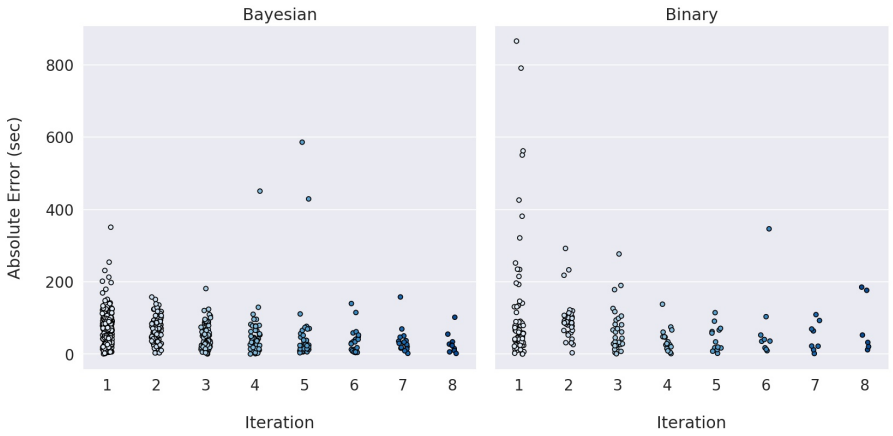


Figure 5.5: **Absolute errors vs. Iteration for Sudoku**. We show the absolute error for each one of the traces. In both experiments, we see how the error gets progressively smaller after each iteration. We also notice that the prior of our approach proposed an initial Sudoku that was closer, on average, to the target goal. In contrast, the Binary search's initial guess of $(81-17)/2$ proved to be too hard for several players.

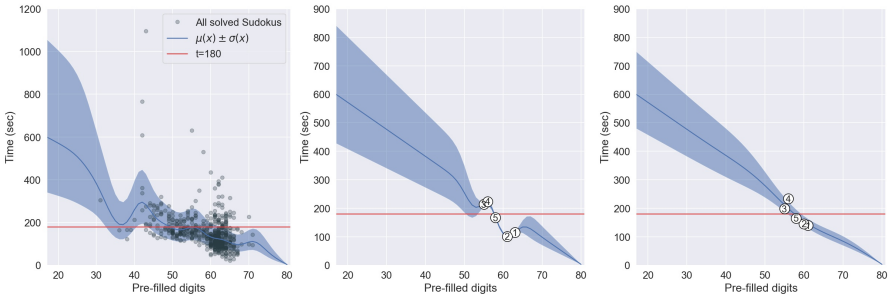
However, diving deeper into the data we realize that this varies depending on the iteration of the content adaption. The first puzzle presented in our approach, i.e. the one specified by our hand-crafted prior, proved to be easy and relatively close to the target for most players (with $\mu_t^{\text{Bay}} \approx 142$ seconds for the first iteration). In sharp contrast, we see that the first proposal of the binary search took players roughly 300 seconds on average to solve, with high variances. We do not see statistical significance when examining the differences after the first iteration, but both approaches indeed work: the average error tends to decrease as iterations go up, as illustrated in Fig. 5.5.

Fig. 5.6a shows the GP posterior resulting from fitting with all correctly solved Sudokus and their respective completion times. Our method mostly proposed *easy* Sudokus, with most of the solved Sudokus (denoted by black dots) starting on the right-hand side of 50 pre-filled cells, and under the 180-second target. Figs. 5.6b and 5.6c show two traces for different users, both taking 5 iterations to find a puzzle with completion time close to our target. For example, the trace shown in Fig. 5.6b is

$$\{(63,115),(61,101),(55,213),(56,223),(58,167)\}.$$

In other words, the framework first proposed puzzles that were too easy (with 63 and 61 pre-filled digits), followed by two puzzles that proved too difficult (55 and 56 pre-filled cells); finally, the optimization process found a good puzzle at 58 hints.

Anecdotally, we found that using linear regression of log-times as a model of the player resulted in predicted times that were not realistic. One example is shown in Fig. 5.9: fitting a linear regression of log-times results in predicting that very sparse Sudokus correspond to low completion times.



(a) (b) (c)

Figure 5.6: **The average Sudoku player, and a couple of traces**. (a) shows the result of fitting a GP with all the traces collected. In a sense, it shows *the average Sudoku player* according to our data. (b) and (c) show two individual traces. In both of these, our framework took 5 iterations to find a Sudoku with the right level of difficulty.

level #s	n_{Bay}	n_{NH}	n_{Rand}	μ_e^{Bay}	μ_e^{NH}	μ_e^{Rand}	H_0^{NH} rejected	H_0^{Rand} rejected
$1 \le i < 5$	130	78	112	3.9± 3.5	5.8± 5.6	7.1± 6.7	yes ($p = 0.01$)	yes ($p = 0.00$)
$5 \le i < 10$	122	82	90	4.4± 5.7	4.9± 5.0	6.3± 6.0	no ($p = 0.51$)	yes ($p = 0.02$)
$10 \le i < 15$	94	59	50	4.1± 4.3	4.4± 6.8	4.7± 3.8	no ($p = 0.75$)	no ($p = 0.42$)
$15 \le i < 20$	73	51	35	4.5± 6.0	3.7± 3.1	6.4± 6.4	no ($p = 0.32$)	no ($p = 0.15$)
$20 \le i < 25$	43	29	27	3.3± 3.2	4.4± 4.9	5.1± 5.4	no ($p = 0.30$)	no ($p = 0.12$)
$25 \le i < 30$	31	18	16	3.2± 2.3	5.2± 6.2	4.0± 3.1	no ($p = 0.20$)	no ($p = 0.40$)
$30 \le i < 35$	25	15	11	3.3± 2.4	4.0± 4.0	4.5± 1.9	no ($p = 0.57$)	no ($p = 0.12$)
All	595	376	360	4.0± 4.7	4.7± 5.3	5.9± 5.7	yes ($p = 0.03$)	yes ($p = 0.00$)

Table 5.2: **Average absolute errors for Dungeon Crawler**. If we aggregate all results for the experiments, we see that our approach (denoted Bayesian) gets an average absolute error that is significantly lower than that of the two baselines (noisy hill-climbing and sampling random levels). This pattern does not necessarily hold for the grouped iterations, where we fail to see any statistical significance.

This speaks to the value of having a Gaussian Process model in which a prior can be specified.

DUNGEON CRAWLER We received 34 unique traces for our proposed framework[12], 21 traces for our hill-climbing baseline, and 30 traces for selecting levels at random.

Table 5.2 shows the average absolute errors, aggregated in groups of 5 for smoothing and ease of presentation. Similarly to our Sudoku experiment, we see that our approach performs better when considering all level-time pairs. However, the margin between ours and the baselines is minimal: about a second against the noisy hill climb, and two seconds against ran-

12 The avid reader might have noticed that this data differs from the one presented in our CoG 2021 contribution (González-Duque, Palm, and Risi, 2021). We left our web application running after presenting our results and managed to store several more playtraces for our Bayesian-based approach.

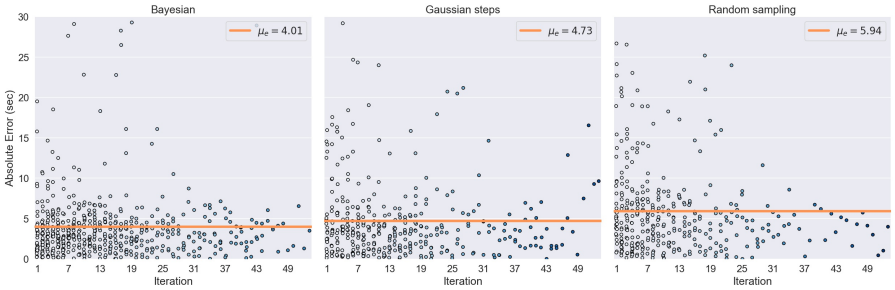


Figure 5.7: **Average error vs. Iteration for Dungeon Crawler**. We show the mean absolute errors per iteration for both our approach (Bayesian) as well as the two baselines (doing noisy hill-climbing by taking Gaussian steps, and sampling levels at random). The aggregated error for each method is highlighted as a horizontal bar, with the exact numerical value shown in the legend. On average, our method performs slightly better than the two baselines according to this metric. A deeper dive into the data shows that, iteration-wise, our model does not necessarily perform better.

dom sampling. This margin is visualized in Fig. 5.7, which plots the average errors for each iteration.[13]

We show the first and eighth iterations of an example playtrace in Fig. 5.8. The more a user plays, the better the surrogate GP approximates their completion time. In its first iteration, the system proposes a level with 5 enemies and a reachability of 11, which takes the user roughly 5 seconds to solve. At the eighth iteration, the optimization finds a level that takes the user 9.6 seconds to solve, with more enemies and a larger distance to the goals.

5.7 DISCUSSION & LIMITATIONS

Our two experiments show different stories: on the one hand, we see that our framework was able to reliably find a Sudoku puzzle that takes three minutes to solve in roughly 7 iterations, maintaining a model of the player's completion time as a by-product. We also noticed how Gaussian Processes allowed for imposing priors and modeling uncertainty, making them more robust than simpler alternatives like greedy decision-making using linear interpolation. Similar results were achieved by the binary search baseline, but only after the first iteration. This speaks to the value of our model's initial guess.

On the other hand, the results for our Dungeon Crawler experiment show, when aggregated, a smaller impact. Our framework showed users levels that took them, on average, an amount closer to the target goal of $t_g = 10$ seconds. However, this difference is small when compared to the average

13 We removed all solved levels that took more than one minute since the levels were designed to be fast-paced. More than a minute of completion time would imply, in our eyes, that the user got distracted.

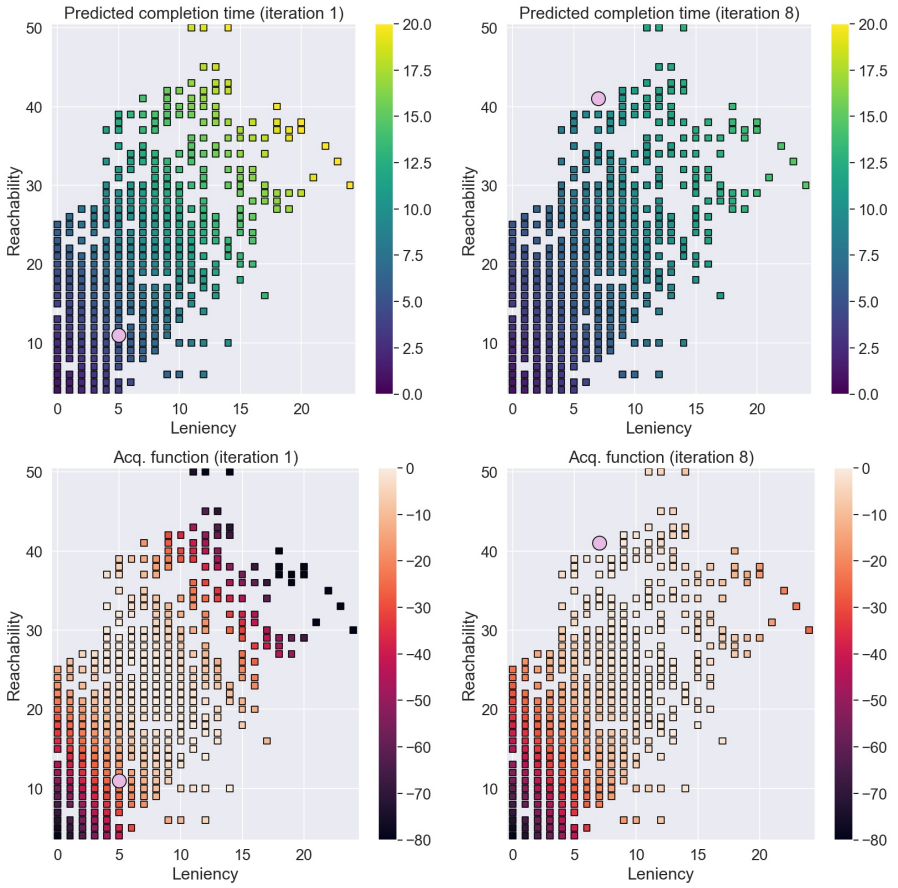


Figure 5.8: **Example playtrace for Dungeon Crawler**. The top row shows the model's prediction for completion time in the first and eighth iterations of one playtrace using our framework. The bottom row shows the acquisition function. We see how the "ideal level" according to our system starts in the middle of the corpus and progressively moves upwards towards more difficult levels. Indeed, the first level proposed was solved in 5.4 seconds, and the eighth took 9.6 seconds (these are highlighted using a pink circle).

errors of the baselines two baselines. More to the point, our method does not perform better if we look at the data at the individual iteration level.

This difference in quantitative results between Sudoku and Dungeon Crawler speaks to the different nature of these games: Sudoku being deterministic, having a one-dimensional encoding and a lowering branching factor, allowed for easier content adaption than Dungeon Crawler. Future research could focus on testing this framework in the interface between these two games: games that are deterministic with a high branching factor, or stochastic games with lowering branching factor.

We now discuss some limitations of our approach and analysis, pointing towards potential future work

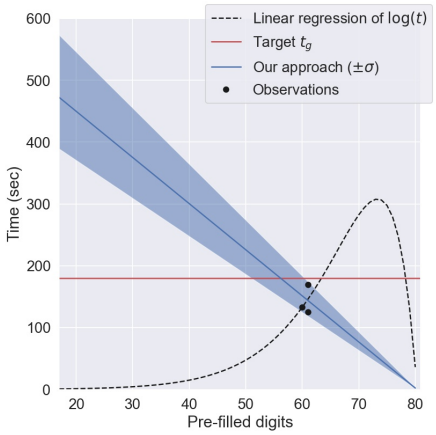


Figure 5.9: **Linear regression vs. GP**. Modeling with linear regression in log-space can result in unrealistic predictions of completion time, unlike a GP with an informative prior. The dashed line (linear regression of log-times) indicates that sparse puzzles are easier than full ones, while the continuous blue line (GP) still predicts sparse levels to be difficult.

SELECTING A GAME REPRESENTATION Our framework relies on a hand-crafted representation for game content. A single game level can be represented in a plethora of ways. Indeed, we could have encoded Sudokus not in terms of their number of pre-filled cells, but rather in terms of its naked singles, naked doubles, etc.; likewise for our dungeon crawler. In Chap. 9 we explore learning this representation automatically using Variational Autoencoders, and we optimize content from these latent, automatically learned representations.

CRAFTING A PRIOR In our analysis, we used hand-crafted priors for both experiments. These leveraged our expert knowledge and assumptions about what impacts the player's completion time. We saw in the Sudoku experiment how much of an impact a good prior can have (see the discussion of Fig. 5.9). Such knowledge might not be available to designers, but alternatives such as bootstrapping with random playtraces or using artificial agents as proxies for building priors (Kristensen, Valdivia, and Burelli, 2020) could be explored in the future.

SAMPLE SIZE We only managed to collect a low amount of playtraces for both experiments. For the Dungeon Crawler, in particular, we analyzed less than 35 traces per method. For a proper analysis to take place, we would need to collect more data to control for noise.

NOISY LEVELS The way the Dungeon Crawler levels were shown was a source of noise: the same level, when played in two different iterations, would result in different enemy movements. In other words, the same level could turn out to be extremely easy in one iteration, and near impossible in the next.

ADDRESSING PLAYER IMPROVEMENT OVER TIME By using the entire playtrace, we are assuming that the players have a *static* skill level. This assumption is often unrealistic, and to address it we could "forget" initial parts of the trace using a rolling window.

5.8 CONCLUSION

In this contribution, we explored the use of Bayesian Optimization for adapting content to players. We developed and tested a framework that models a metric of the player (completion time in both our experiments), and leverages this model to propose new content, aiming to present tasks that measured close to a certain target t_g in the metric being evaluated.

This framework was tested on two games: Sudoku, and a Dungeon Crawler clone of *The Legend of Zelda*. For Sudoku, our system was able to find a puzzle of around 3 minutes to solve in roughly 7 iterations, maintaining a model of the player's completion time in the process. This performance matched that of simpler baselines that do not have player models as by-products. For the Dungeon Crawler, the results show only a slight improvement with high variance against simpler baselines. Our framework, then, is an alternative for exploring the design spaces of a given game for a piece of content that matches certain criteria specified by the developers.

Part III

APPLICATIONS OF DEEP GENERATIVE MODELS AND DIFFERENTIAL GEOMETRY

6

AN INTRODUCTION TO DEEP GENERATIVE MODELS IN VIDEO GAMES

The third part of the thesis deals with applications of Deep Generative Models (DGMs): methods for generating novel samples of the distribution of a given dataset. This chapter gives an introduction to three families of DGMs: Autoregressive Models (ARMs), Generative Adversarial Networks (GANs), and Variational Autoencoders (VAEs). The main focus is placed on the latter since it is the main model used in the experiments described in the following chapters.

This chapter starts with a general introduction to generative modeling, followed by an introduction to the three types of models discussed above. Then, the use of these models in video games is surveyed. Finally, we discuss an issue that arises when using DGMs for modeling content that is expected to be functional: *the content generated by these models is not guaranteed to work* (e.g. the video game levels sampled from these models may not be playable (Liu et al., 2020, Sec. 3), (Summerville et al., 2018, Sec. 4.A)).

The main references for this chapter are Jakub Tomczak's book *Deep Generative Modeling* (Tomczak, 2022), the comparative survey on DGMs by Bond-Taylor et al. (Bond-Taylor et al., 2022), the survey on using Machine Learning on Procedural Content Generation (PCGML) by Summerville et al. (Summerville et al., 2018), and the survey on Deep Learning for PCG by Liu et al. (Liu et al., 2020).[1]

6.1 WHAT IS GENERATIVE MODELING?

Generative models approximate the distribution of a training set. In other words, these models assume that the dataset in data space $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} \subseteq \mathbb{R}^D$ (e.g. images of faces, molecules, video game levels) is sampled from a probability distribution $p(\boldsymbol{x})$, and the goal of the model is to approximate it.

This chapter covers three out of a plethora of modeling choices for the data's distribution. In particular, we focus on *deep* generative models, models in which the distribution $p(\boldsymbol{x})$ is approximated using artificial neural networks. Beyond ARMs, GANs and VAEs, which are our concern for the rest of the chapter, there are three other main families of DGMs:

1 We would like to argue that there is room for a survey of the use of DGMs in games since, as this chapter hopefully shows, it is a growing field as of itself. Comparing the affordances of DGMs would bring value to the PCGML community.

FLOW-BASED MODELS (FBMS) leverage invertible transformations to map a known distribution (e.g. Gaussian noise) into the distribution of the data. When a random variable is transformed using an invertible map f, the distribution after the transformation can be computed in closed form using the *change of variables* rule (Tomczak, 2022, Chap. 3). In flow-based models, invertible neural networks learn to transform a simple distribution into $p(\boldsymbol{x})$ and backwards.

ENERGY-BASED MODELS (EBMS) use the *Boltzmann distribution* to approximate $p(\boldsymbol{x})$. This distribution is given by $\exp(E_\theta(\boldsymbol{x}))/Z_\theta$, where $E_\theta(\boldsymbol{x})$ is an energy function parametrized by a neural network (Tomczak, 2022, Chap. 6), giving the distribution plenty of flexibility.

DIFFUSION-BASED MODELS (DDGMS) learn to transform the distribution $p(\boldsymbol{x})$ into e.g. Gaussian noise by following a *diffusion process* which gradually adds noise and simultaneously learns to denoise (Tomczak, 2022, Sec. 4.5.3).

These three families of DGMs are ripe for application in video game content since they are also capable of modeling discrete distributions like the ones that govern tile-based levels (Hoogeboom et al., 2019; 2021); to the best of our knowledge, they have not found application by the PCGML community yet.

6.2 AUTOREGRESSIVE MODELS

One way to approximate $p(\boldsymbol{x})$ when $\boldsymbol{x} = (x_1, \ldots, x_L)$ is sequential (e.g. a sentence, or an image) is to factor $p(\boldsymbol{x})$ as

$$p(\boldsymbol{x}) = \prod_{l=1}^{L} p(x_l | x_1, \ldots, x_{l-1}) = \prod_{l=1}^{L} p(x_l | x_{<l}), \tag{6.1}$$

followed by modeling these conditional distributions using a neural network. The family of models that consider this factorization is called *autoregressive*.

Two other standard autoregressive models are PixelCNN (Oord et al., 2016) and PixelRNN (Oord, Kalchbrenner, and Kavukcuoglu, 2016), which account for a bigger "context" than just the previous recurrent cell.

By modeling $p(\boldsymbol{x})$ directly, autoregressive models have direct access to the likelihood of the data (i.e. these models can be used to determine whether a given piece of content is in-distribution or not). Unfortunately, the use of recurrent units makes sampling from autoregressive models expensive (Bond-Taylor et al., 2022, Table 1.).

6.3 GENERATIVE ADVERSARIAL NETWORKS

Proposed by Goodfellow et al. in 2014, Generative Adversarial Networks (GANs) learn to sample from the distribution $p(\boldsymbol{x})$ using a two-player game: a generator Gen_θ and a discriminator Disc_ϕ, parametrized by neural networks with parameters θ and ϕ, are trained such that (i) the generator creates content that is misclassified as "real" by the discriminator, and (ii) the discriminator is able to correctly identify the content generated by Gen as artificial, and the "real" content.

More precisely, $\text{Disc}_\phi : \mathbb{R}^D \to \{0,1\}$ is a classifier that is fed both samples from the dataset and content artificially generated by Gen_ϕ, and its goal is to correctly identify the ones that come from the dataset. The generator $\text{Gen} : \mathbb{R}^d \to \mathbb{R}^D$ learns to transform random noise $\boldsymbol{z} \in \mathbb{R}^d$ into samples $\tilde{\boldsymbol{x}} \in \mathbb{R}^D$ (where $d < D$). These two networks are trained to optimize the following value (Goodfellow et al., 2014):

$$\min_{\text{Gen}_\theta} \max_{\text{Disc}_\phi} \left(\mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x})} [\log \text{Disc}_\phi(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim N(\boldsymbol{0}, \boldsymbol{I}_d)} [\log(1 - \text{Disc}_\phi(\text{Gen}_\theta(\boldsymbol{z})))] \right). \tag{6.2}$$

In their vanilla form, GANs are notoriously unstable and difficult to train (Bond-Taylor et al., 2022). Alternative loss functions have been proposed, with the promise of more stability during training (Arjovsky, Chintala, and Bottou, 2017).

GANs have found application in several domains and were considered the most competitive alternative when it came to generating realistic (unconditioned) samples of faces, animals and objects (Karras et al., 2020). However, since 2021 diffusion models are considered the state-of-the-art for generating realistic samples of images (Dhariwal and Nichol, 2021).

To summarize, GANs are able to synthesize content with high quality, and have found several applications including the generation of video game content; the sampling process is fast (involving only a feed-forward pass through Gen_θ), but fitting these networks to a dataset is known to be unstable and difficult.

6.4 VARIATIONAL AUTOENCODERS

Variational Autoencoders (VAEs) (Kingma and Welling, 2014; Rezende, Mohamed, and Wierstra, 2014) are a probabilistic interpretation of the classical Autoencoders (Rumelhart and McClelland, 1987), a neural network with a bottleneck. Since VAEs are the main model used in our methods and contributions, we explain them in detail.

6.4.1 *Autoencoders*

An autoencoder is composed of two subnetworks: an encoder $\text{enc}_\phi : \mathbb{R}^D \to \mathbb{R}^d$ and a decoder $\text{dec}_\theta : \mathbb{R}^d \to \mathbb{R}^D$ (with trainable parameters θ and ϕ). In

the original setting (Ballard, 1987; Schmidhuber, 2015) they are trained to learn an identity mapping that passes through the d-dimensional bottleneck:

$$\text{Loss}(\theta, \phi; \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}) = \sum_{n=1}^{N} \|\boldsymbol{x}_i - \text{dec}_\theta(\text{enc}_\phi(\boldsymbol{x}_i))\|^2. \tag{6.3}$$

Autoencoders learn a low-dimensional *latent representation* $\boldsymbol{z} = \text{enc}_\phi(\boldsymbol{x})$. These are considered deterministic quantities in the AE setting, but VAEs interpret them probabilistically by learning their approximate posterior distributions.

6.4.2 *A distribution over latent variables*

The next three subsections summarize „Auto-Encoding Variational Bayes“ (Kingma and Welling, 2014). If the reader is already familiarized with VAEs, they can skip to Sec. 6.4.6.

Variational Autoencoders were proposed by Kingma and Welling (2014) as an example of efficient variational inference in *latent variable models*: graphical models in which the data $\boldsymbol{x} \in \mathbb{R}^D$ depends on latent representations $\boldsymbol{z} \in \mathbb{R}^d$. The authors formulate the following probabilistic model:

$$\begin{aligned} \boldsymbol{z} &\sim p(\boldsymbol{z}) &&\text{: a prior over the latent variables.} \\ \boldsymbol{x} &\sim p_\theta(\boldsymbol{x}|\boldsymbol{z}) &&\text{: a likelihood of the data, depending on } \boldsymbol{z}. \\ \boldsymbol{z} &\sim q_\phi(\boldsymbol{z}|\boldsymbol{x}) &&\text{: an approximate posterior of } \boldsymbol{z} \text{ given } \boldsymbol{x}. \end{aligned}$$

The distributions $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ are parametrized using neural networks and can be seen as probabilistic interpretations of the encoder and decoder respectively.

The reason why the authors *approximate* the posterior distribution of the latent codes given data is that the actual posterior is *intractable* in most cases. Indeed, the posterior $p(\boldsymbol{z}|\boldsymbol{x})$ is given by $p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})/p(\boldsymbol{x})$, making the evidence $p(\boldsymbol{x})$ a necessary part of the computation. The word *variational* in the name refers to approximating this distribution using another one with a known form, e.g. optimizing the parameters of a multivariate Gaussian to minimize a notion of “distance” (Blei, Kucukelbir, and McAuliffe, 2017, Sec. 2).

To know how to optimize the parameters ϕ and θ of the encoder and decoder, Kingma and Welling derive a lower bound of the evidence $p(x)$. This lower bound, which we explain in the next subsection, can be easily computed and differentiated using autodifferentiation tools like `torch` (Paszke et al., 2019).

6.4.3 *The Evidence Lower Bound (ELBO)*

Recall that a marginal distribution like $p(\boldsymbol{x})$ can be written as $\int p_\theta(\boldsymbol{x}|\boldsymbol{z})\,p(\boldsymbol{z})\mathrm{d}\boldsymbol{z}$ as per the sum and product rule (Bishop, 2006, pg. 14). With this in mind, the authors derive a lower bound on $\log p(\boldsymbol{x})$ as follows:

$$\log p(\boldsymbol{x}) = \log\left(\int p_\theta(\boldsymbol{x}|\boldsymbol{z})\,p(\boldsymbol{z})\mathrm{d}\boldsymbol{z}\right) \tag{6.4}$$

$$= \log\left(\int \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})\,p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\,q_\phi(\boldsymbol{z}|\boldsymbol{x})\mathrm{d}\boldsymbol{z}\right) \tag{6.5}$$

$$= \log\left(\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})\,p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right]\right) \tag{6.6}$$

$$\geq \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})\,p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right)\right] \tag{6.7}$$

$$= \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z})}\right)\right], \tag{6.8}$$

where Eq. (6.4) holds because of the sum and product rules of probability, in Eq. (6.5) we multiplied and divided by $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, Eq. (6.6) follows from the definition of expectation, Eq. (6.7) is an application of Jensen's inequality for concave functions, like log, and finally Eq. (6.8) is an application of properties of logarithms.

The second summand in the right-hand side of Eq. (6.8) is known as the *Kullback-Leibler divergence* (KL), denoted more generally by

$$\mathrm{KL}(q(\boldsymbol{z})||p(\boldsymbol{z})) = \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z})}\left[\log\left(\frac{q(\boldsymbol{z})}{p(\boldsymbol{z})}\right)\right]. \tag{6.9}$$

KL measures *how similar* two distributions are, and can be thought of as something similar to a distance. The key difference between divergences and distances is that divergences are not symmetric in their inputs, while distances are. Moreover, divergences do not necessarily satisfy the triangle inequality.

For most distributions of interest, the KL divergence can be computed analytically; otherwise, it can be approximated by averaging the log-quotient after taking samples from $\boldsymbol{z}\sim q(\boldsymbol{z})$.[2]

Substituting the definition of KL in Eq. (6.8), we arrive at the *Evidence Lower Bound* (ELBO) objective function $\mathscr{L}$, which is used for optimizing the parameters ϕ and θ:

$$\mathscr{L}(\theta,\phi;\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})). \tag{6.10}$$

The first term of the ELBO is an expected *reconstruction error*, a probabilistic generalization of the original loss function specified for Autoencoders in Eq. (6.3).[3] The choice of likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ depends on the nature of the

2 This is known as *Monte Carlo Integration*.

3 Indeed, if we remove the expectation, consider a single $\boldsymbol{z} = \mathrm{enc}(\boldsymbol{x})$, and use a Gaussian likelihood, we arrive at Eq. (6.3) exactly.

data. If what is being modeled is known to be real numbers, then choosing a Gaussian likelihood is customary; if the variables are probabilities between 0 and 1 (like in the case of modeling bounded values like the pixels of MNIST digits), a Bernoulli likelihood would be more appropriate. Each of these results in a different reconstruction error (the usual sum-of-squares for the Gaussian likelihood, and binary cross-entropy for the Bernoulli).

The second term is usually described as a *regularization*, which shapes the approximate posterior to be similar to the prior $p(\boldsymbol{z})$, which is usually taken to be a multivariate Gaussian with identity covariance $N(\mathbf{0}, \boldsymbol{I}_d)$.

6.4.4 *The reparametrization trick*

Since maximizing the evidence directly is not tractable, Kingma and Welling maximize the ELBO (i.e. a lower bound of it) in Eq. 6.10. Computing the gradient of the ELBO w.r.t the parameters ϕ is not straightforward in its current form, since we are taking expectations w.r.t the distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$.

To explain how it is feasible to differentiate the ELBO with respect to ϕ, we focus on the case where $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is a multivariate Gaussian with parameters $N(\boldsymbol{\mu}_\phi(\boldsymbol{x}), \text{diag}(\boldsymbol{\sigma}_\phi(\boldsymbol{x}))^2)$, where $\boldsymbol{\mu}_\phi(\boldsymbol{x})$ and $\boldsymbol{\sigma}_\phi(\boldsymbol{x})$ are d-dimensional vectors and $\text{diag}(\boldsymbol{\sigma}_\phi(\boldsymbol{x})^2)$ is a matrix with $\boldsymbol{\sigma}_\phi(\boldsymbol{x})^2$ in its diagonal.

In this scenario, samples $\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})$ can be re-written as

$$\boldsymbol{z} = \boldsymbol{\mu}_\phi(\boldsymbol{x}) + \boldsymbol{\varepsilon} \odot \boldsymbol{\sigma}_\phi(\boldsymbol{x})^2, \;\; \boldsymbol{\varepsilon} \sim N(\mathbf{0}, \boldsymbol{I}_d). \tag{6.11}$$

This is known as *the reparametrization trick*: it allows for reparametrizing the expectation in the ELBO (Eq. 6.10) in this specific example as

$$\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] = \mathbb{E}_{\boldsymbol{\varepsilon}\sim N(\mathbf{0},\boldsymbol{I}_d)}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{\mu}_\phi(\boldsymbol{x}) + \boldsymbol{\varepsilon} \odot \boldsymbol{\sigma}_\phi(\boldsymbol{x})^2)\right]. \tag{6.12}$$

After this reparametrization, it is straightforward to compute the gradient ∇_ϕ of these expectations.

To summarize, the gradient of the ELBO can be computed for distributions that allow to re-write the sampled latent variables $\boldsymbol{z}$ as a deterministic function that depends on *other* random variables that do not involve the parameters. Beyond the multivariate Gaussian, several other distributions can and have been studied in latent space (Davidson et al., 2018; Oord, Vinyals, and kavukcuoglu, 2017).

6.4.5 *Variational Autoencoders*

What is discussed in earlier subsections is a general method for doing a variational approximation of the distributions when facing a latent variable

model. Kingma and Welling discuss Variational Autoencoders as a particular instance of this framework, where

$\boldsymbol{z} \sim p(\boldsymbol{z}) = N(\mathbf{0}, \boldsymbol{I}_d)$, i.e. a Normal prior.

$\boldsymbol{x} \sim p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \text{dec}(\boldsymbol{z};\theta)$, a likelihood approximated using a decoder.

$\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}) = N(\boldsymbol{\mu}_\phi(\boldsymbol{x}), \text{diag}(\boldsymbol{\sigma}_\phi(\boldsymbol{x})^2))$, where $\text{enc}(\boldsymbol{x};\phi) = [\boldsymbol{\mu}_\phi(\boldsymbol{x}), \boldsymbol{\sigma}_\phi(\boldsymbol{x})]$.

The encoder and decoder are neural networks whose parameters can be optimized by maximizing the ELBO. Since we are approximating the posterior using a multivariate Gaussian, and these allow for reparametrization, computing the gradients of the ELBO is straightforward. Moreover, the KL divergence between the prior $p(\boldsymbol{z})$ and the approximate posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ has closed analytical form, and can be evaluated without resorting to sampling and averaging.

Depending on the likelihood, we will think about the decoder as returning the *parameters* of the distribution. For example, in the case where $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is a Gaussian, the decoder will output $\text{dec}(\boldsymbol{z};\theta) = [\boldsymbol{\mu}_\theta(\boldsymbol{z}), \boldsymbol{\sigma}_\theta(\boldsymbol{z})]$, both of these being vectors in data space $\mathbb{R}^D$.

6.4.6 *An example: MNIST(1)*

This subsection presents a first example of a Variational Autoencoder trained on only the digits corresponding to class "1" in the MNIST dataset (Lecun et al., 1998).[4] This dataset is known to have a curved, non-convex structure in latent space even when using linear dimensionality reduction techniques like Principal Component Analysis (PCA).

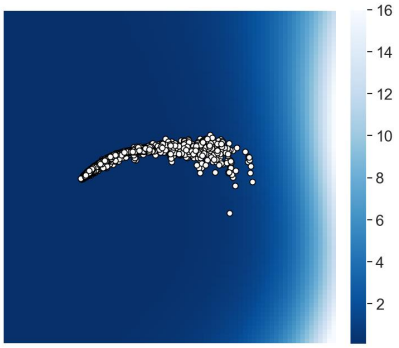


Figure 6.1: **Latent space of MNIST(1)**. The color map corresponds to the value of $\boldsymbol{\sigma}_\theta(\boldsymbol{z})$. Training a VAE results in unreliable uncertainty estimates, with areas far from the training codes having low variance.

We train a VAE[5] with a 2-dimensional latent space that decodes to a multivariate Gaussian in data space $\mathbb{R}^{28\times 28}$. Fig. 6.1 shows the resulting latent space. The latent codes correspond to evaluating $\boldsymbol{\mu}_\phi(\boldsymbol{x})$ in all the training data.

Unfortunately, *VAEs lack good uncertainty quantification* (Rybkin, Daniilidis, and Levine, 2021). By this, we mean that the standard deviations learned by the network encoder and decoder do not reflect any structure in the data, as can be seen in Fig. 6.1. Arvanitidis, Hansen, and Hauberg (2018)

4 The code for this example is made available in https://github.com/miguelgondu/examples_in_thesis.

5 The specific details about hidden layers and training hyperparameters can be found in Appendix. A.3.1.

propose a method for properly "calibrating" the uncertainties of VAEs with Gaussian decoders, and we discuss it using this same example in Sec. 7.2.1. Examples of latent variable models that properly quantify uncertainty include GPLVMs (Lawrence, 2003), and modifications of VAEs (Miani et al., 2022; Rybkin, Daniilidis, and Levine, 2021; Skafte, Jørgensen, and Hauberg, 2019).

6.5 VAES ON DISCRETE INPUTS: THE CATEGORICAL LIKELIHOOD

This thesis studies the application of VAEs for learning a continuous representation of discrete, tile-based video game levels. Since this is our primary focus, this section introduces the relevant terminology and methods.

To model such discrete content, we choose the *Categorical distribution* (denoted Cat) as the data likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. The Categorical models discrete sequences $\boldsymbol{x} = (x_1, \dots, x_L)$, where each element of the sequence is one of C possible "tokens" (or tiles, in the case of video game levels). These set of tiles are called a *vocabulary*, and are denoted by $V = \{t_1, \dots, t_C\}$.

To make this notation more concrete, consider the example of a level from SMB. The tokens correspond to the 11 different tiles shown in Table 6.1. In our setup, a data point $\boldsymbol{x} = (x_1, \dots, x_{14\times 14})$ corresponds to a 14×14 level where each x_l is a tile in V.

Tiles	Token
	t_1 = X
	t_2 = S
empty	t_3 = -
	t_4 = ?
	t_5 = Q
	t_6 = E
	t_7 = <
	t_8 = >
	t_9 = [
	t_{10} =]
	t_{11} = o

Table 6.1: **Vocabulary in SMB**

Instead of dealing with discrete data, these tokens are transformed into their *one-hot encoding*: probability vectors $\boldsymbol{p}_i = (p_{i,1}, \dots, p_{i,C})$ defined by $p_{i,c} = [x_i = t_c]$, where $[a = b] = 1$ if $a = b$ and 0 otherwise.[6] Fig. 6.2 illustrates an example vector in an SMB level.

With this notation, we can introduce the Categorical distribution more formally. For an individual x, which takes values in a vocabulary with C tokens, $x \sim \text{Cat}(x|(p_1, \dots, p_C))$ means that the probability of x being t_c is given by p_c. Mathematically:

$$\text{Prob}[x = t_c] = p_c. \tag{6.13}$$

When it comes to entire sequences $\boldsymbol{x} = (x_1, \dots, x_L)$, each x_l is modeled independently using a Categorical. In other words, each x_l is distributed according to $\text{Cat}(x_l|[p_{l,1}, \dots, p_{l,C}]) =: \text{Cat}(x_l|\boldsymbol{p}_l)$. Under this independence assumption, we write $\text{Cat}(\boldsymbol{x}|\{\boldsymbol{p}_1, \dots, \boldsymbol{p}_L\})$ to denote the fact that each x_l is distributed according to the probabilities $\boldsymbol{p}_l$.

6 $[a = b]$ is known as *Iverson's bracket*.

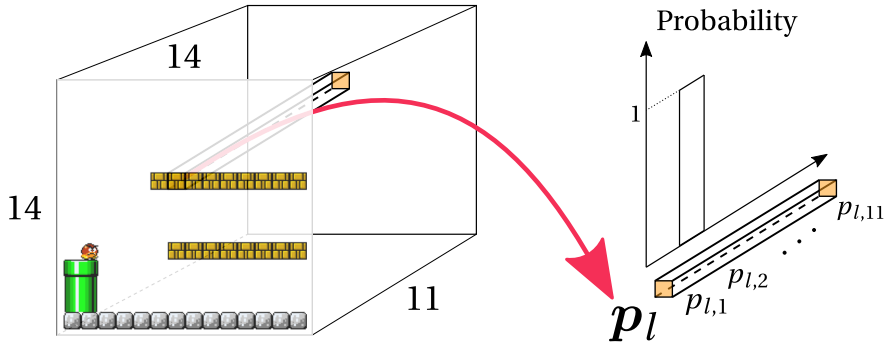


Figure 6.2: **Modeling discrete data using probability vectors**. We show the construction of the one-hot encoding for an example in SMB. Highlighting a tile $x_l = t_2$ (i.e. a breakable stone), a probability vector $\boldsymbol{p}_l$ is constructed such that $p_{l,2} = 1$ and the rest are 0. After this transformation, each level $\boldsymbol{x}$ becomes a tensor with 3 dimensions, where the first two correspond to the positions l and the last one corresponds to which tile the level should decode to.

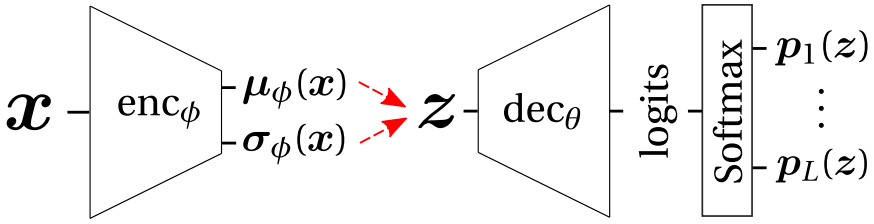
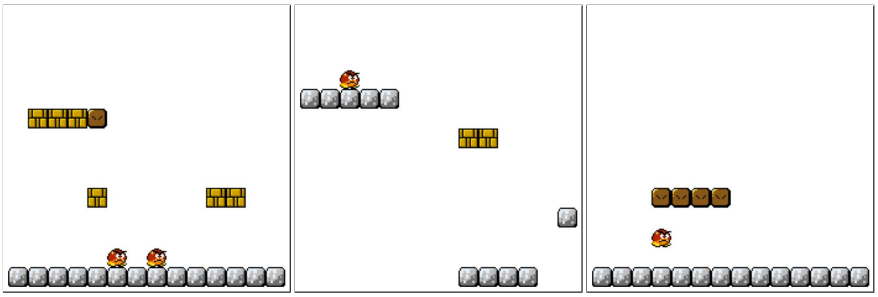


Figure 6.3: **Variational Autoencoder with Categorical Likelihood**. This diagram shows how a given discrete sequence $\boldsymbol{x} \in \mathbb{R}^L$ is transformed into the parameters of the approximate Gaussian posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ with parameters $\boldsymbol{\mu}_\phi(\boldsymbol{x})$ and $\boldsymbol{\sigma}_\phi(\boldsymbol{x})^2$. This distribution is sampled (denoted with red dashed arrows), outputting latent codes $\boldsymbol{z} \in \mathbb{R}^d$ that get transformed via the decoder into *logits*, unnormalized log-probabilites which, after passing through a Softmax activation, transform into probability vectors, one for each x_l, with $l = 1, \ldots, L$.



(a) (b) (c)

Figure 6.4: **Three examples from SMB**. After our postprocessing of the original SMB levels present in the Video Game Level Corpus (VGLC), we store a dataset of 2713 levels of shape 14×14. This figure shows three of these selected at random.

Circling back to VAEs, the goal is to approximate this likelihood using a decoder. The latent variable model is setup as follows:

$$\begin{aligned}
&\boldsymbol{z} \sim p(\boldsymbol{z}) = N(\mathbf{0}, \boldsymbol{I}_d) \\
&\boldsymbol{x} \sim p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \text{Cat}(\boldsymbol{x}|\{\boldsymbol{p}_1, \dots, \boldsymbol{p}_L\}), \text{where } \text{dec}_\theta(\boldsymbol{z}) = [\boldsymbol{p}_1(\boldsymbol{z}), \dots, \boldsymbol{p}_L(\boldsymbol{z})], \\
&\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}) = N(\boldsymbol{\mu}_\phi(\boldsymbol{x}), \text{diag}(\boldsymbol{\sigma}_\phi(\boldsymbol{x})^2)), \text{where } \text{enc}_\phi(\boldsymbol{x}) = [\boldsymbol{\mu}_\phi(\boldsymbol{x}), \boldsymbol{\sigma}_\phi(\boldsymbol{x})].
\end{aligned} \tag{6.14}$$

Using the Softmax activation function allows for modeling probability vectors like the $\boldsymbol{p}_l(\boldsymbol{z})$ discussed above. Fig. 6.3 shows a diagram of how such a VAE could be structured, and it highlights two important terms:

- **Logits** are the unnormalized log-probabilities, which take values in all the real numbers. They are the output of the final linear layer before the Softmax activation function.
- **Probs** are the probability vectors $\boldsymbol{p}_l(\boldsymbol{z})$, i.e. the result of applying Softmax to the logits.

Tools like `torch.distributions` allow for specifying Categorical distributions in terms of either the logits or the probs.[7]

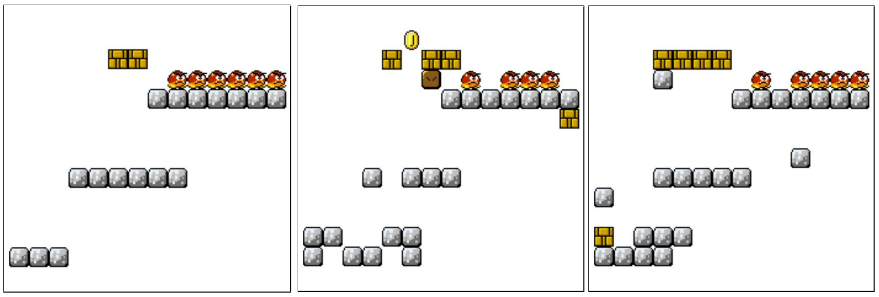
6.6 EXAMPLES OF DISCRETE VAES

This section describes two examples of discrete VAEs: One trained on levels from SMB, and another one trained on levels from The Legend of Zelda. We release the first example in an open-source implementation.[8] These implementations can be easily modified to replicate the other experiment.

6.6.1 *Super Mario Bros levels*

In the *Video Game Level Corpus* (VGLC) (Summerville et al., 2016), the levels from the original Super Mario Bros (SMB) are available as `.txt` files with

7 https://pytorch.org/docs/stable/distributions.html#categorical
8 https://github.com/miguelgondu/minimal_VAE_on_Mario



(a) `argmax` (b) 2 samples from $p_\theta(\boldsymbol{x}|\boldsymbol{z})$

Figure 6.5: **Taking tiles with maximum probability vs. sampling**. After sampling a $\boldsymbol{z}$ at random from the prior, Fig. 6.5a shows the result of considering the tiles that maximize the probability per tile (i.e. taking the `argmax`) in $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. Fig. 6.5b shows two samples from $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, sampling from the Categorical distributions defined by the probability vectors $\{\boldsymbol{p}_1(\boldsymbol{z}),\dots,\boldsymbol{p}_{14\times14}(\boldsymbol{z})\}$ given by the decoder. These probability vectors are visualized, per class, in Fig. 6.6.

the encoding presented in Table 6.1.[9] These are post-processed by sliding a 14×14 window arriving at 2713 levels, which are then transformed to their one-hot encoding. Examples of these 14×14 levels are shown in Fig. 6.4.

We train a VAE[10] with the encoder and decoder specified in Eq. (6.14), using a 2-dimensional latent space. In this example, $L = 14\times14$ and $C = 11$.

After training, decoding a given $\boldsymbol{z}\in\mathbb{R}^d$ results in 14×14 Categorical distributions $p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \text{Cat}(\boldsymbol{x}|\{\boldsymbol{p}_1(\boldsymbol{z}),\dots,\boldsymbol{p}_L(\boldsymbol{z})\})$, one for each tile in a level. Fig. 6.6 shows a heatmap for each one of the probabilities per class in a given decoded sample.

Levels can be reconstructed by either sampling from $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, or by taking the maximum probability per tile. Fig. 6.5 shows examples of each. Unless otherwise stated, we generate levels **by taking the maximum probability per tile** instead of sampling. This applies to the other examples presented in this section, as well as the experiments involving Categorical VAEs.

We can further explore the latent space by decoding an evenly-spaced grid in the $[-5,5]^2$ square. Given a grid size G, a grid of latent codes is constructed by considering G equally-spaced points in the x-axis, as well as the y-axis. This results in a collection of G^2 points $\{(-5,-5),\dots,(5,5)\}$ in latent space. Decoding an evenly-spaced grid like this allows for visualizing the latent space. Fig. 6.7 schematizes this construction and shows the resulting latent space for the running SMB example.

9 https://github.com/TheVGLC/TheVGLC/tree/master/Super%20Mario%20Bros/Processed

10 The training details and model hyperparameters are available in Appendix. A.3.2

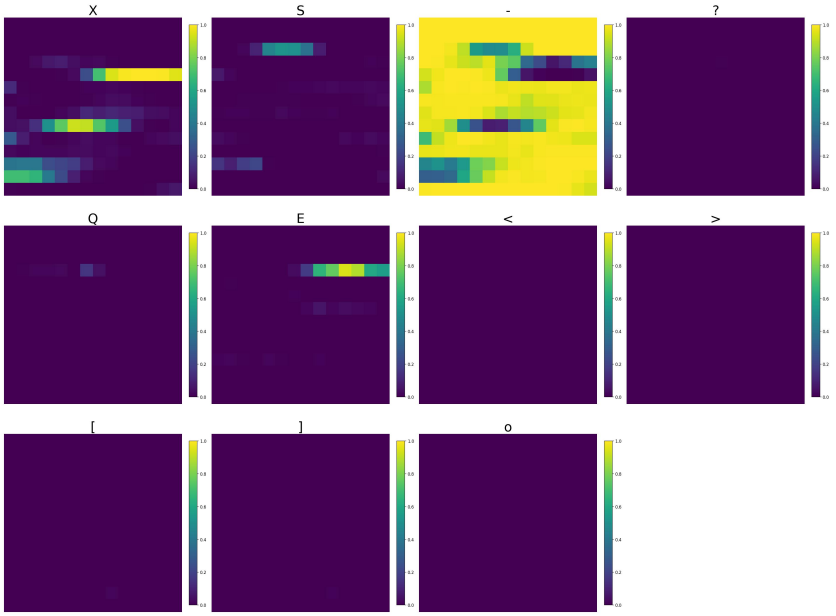


Figure 6.6: **Probability vectors for a sampled z.** Using the same latent code from Fig. 6.5, this figure illustrates the probabilities $p_{\cdot,c}$ for all tiles t_c in the vocabulary of SMB (Table 6.1). The dominant probability is "empty space" -, followed by some ground tiles in the pattern of a platform ladder in X. Enemies E have a high probability of occurring above the last step of the ladder. Fig. 6.5a shows the resulting level from taking, for each position, the token with maximum probability, and Fig. 6.5b shows the result of sampling from the probability vectors.

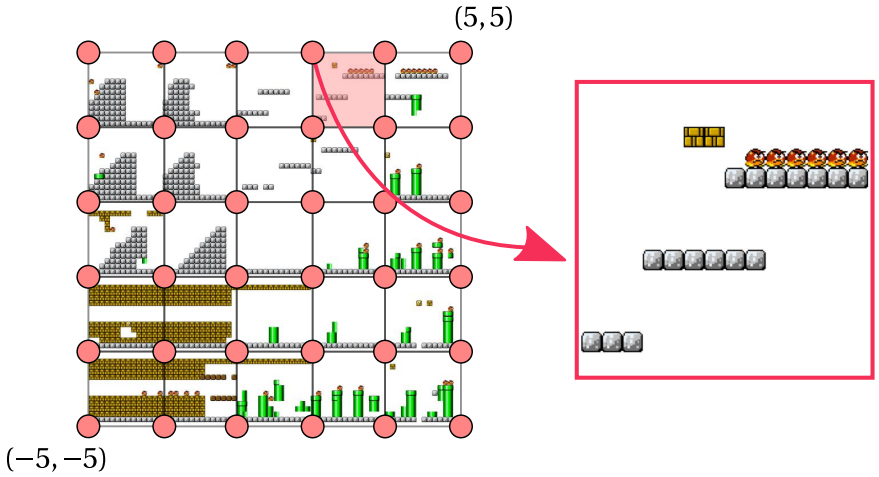


Figure 6.7: **Visualizing latent space with a grid.** We visualize 2-dimensional latent spaces using an evenly-spaced grid of $G \times G$ points in a region of latent space (like the $[-5,5]^2$ square), plotting the decoded images side-to-side. This figure schematizes this construction for $G = 5$ and highlights the region that corresponds to the level presented in Fig. 6.5a. A version of this grid with $G = 10$ is presented in Fig. 9.1a.

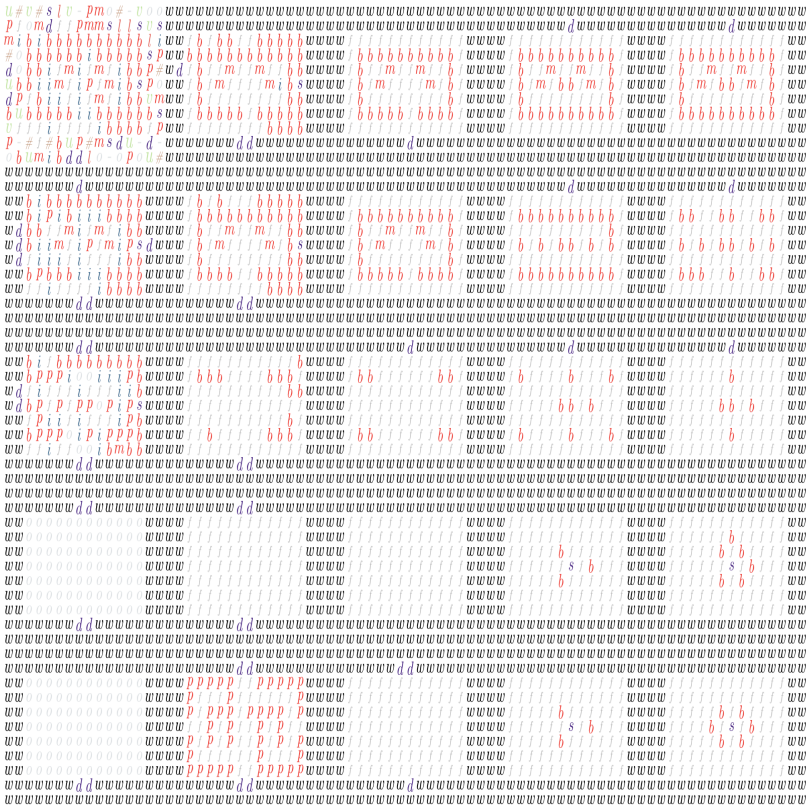


Figure 6.8: **A grid of levels in latent space (Zelda)**. This Fig. shows a grid of $G = 5$ levels in the $[-1, 4]^2$ square in latent space (Fig. 6.7). Levels tend to have incomplete doors, and falling away from the support of the data (e.g. the level in the top-left) results in levels that do not correspond to the distribution.

6.6.2 *The Legend of Zelda levels*

The VGLC also contains levels from the *The Legend of Zelda*[11]. These are available as `.txt` files as well and can be post-processed to extract 237 individual rooms, which are 11×16 in shape.[12] The vocabulary is described in Table A.2 in the appendix, with passable blocks highlighted with "(p)".

A VAE with a 2-dimensional latent space trained[13] on this dataset results in the latent space visualized in Fig. 6.8, and the result of decoding 3 random latent codes $\boldsymbol{z} \sim p(\boldsymbol{z})$ is shown in Fig. 6.9.

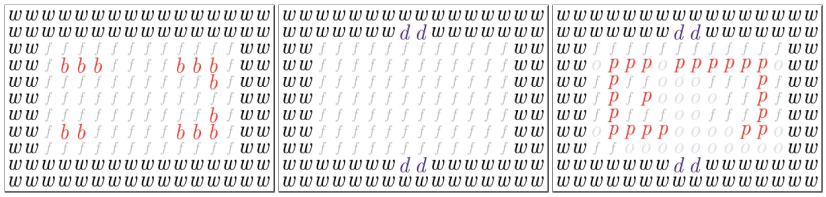
6.7 RELATED WORK: DGMS IN GAMES

This section briefly surveys the use of DGMs in video games.

11 https://github.com/TheVGLC/TheVGLC/tree/master/The%20Legend%20of%20Zelda/Processed

12 This has already been implemented by Schrum, Volz, and Risi. https://github.com/schrum2/GameGAN/tree/master/data/VGLC/Zelda/Processed.

13 See training details in Appendix A.3.3.



(a) (b) (c)

Figure 6.9: **Three examples from Zelda's latent space**. We show three examples of levels decoded from the latent space of Zelda, which come from sampling latent codes at random.

AUTOREGRESSIVE MODELS A first example of an autoregressive model consists of using Recurrent Neural Networks (RNNs) to account for the dependence between x_l and $x_{<l}$. Summerville and Mateas (2016) used Long Short-Term Memory (LSTM) recurrent units to generate levels from SMB. LSTMs have also been used to blend tile-based game levels like SMB and Kid Icarus (Sarkar and Cooper, 2018). Recently, LSTM-based generators have been benchmarked against other DGMs in the game Sokoban (Zakaria, Fayek, and Hadhoud, 2023).

To the best of our knowledge, PixelCNNs and PixelRNNs (alongside their contemporary alternatives (Bond-Taylor et al., 2022)), have not yet been applied to video game content.

GANS In the specific case of video game content, GANs have been used for tile-based content like SMB (Volz et al., 2018), Kid Icarus (Sarkar, Yang, and Cooper, 2019), Zelda (Schrum et al., 2020), several games inside the General Video Game AI framework (Irfan, Zafar, and Hassan, 2019; Kumaran, Mott, and Lester, 2019), Sokoban (Zakaria, Fayek, and Hadhoud, 2023), Mega Man (Capps and Schrum, 2021), and Super Mario Kart (Awiszus, Schubert, and Rosenhahn, 2020) as well levels for DOOM (Giacomello, Lanzi, and Loiacono, 2018).

Several more applications can be found in (Liu et al., 2020, Sec. 4.4), including learning from a single example in both SMB and Minecraft (Awiszus, Schubert, and Rosenhahn, 2020; 2021), and bootstrapping playable content from a GAN trained on Zelda (Rodriguez Torrado et al., 2020). They have also been used as a way to represent content in Reinforcement Learning schemes, aimed at adapting content (Shu, Liu, and Yannakakis, 2021). Interestingly, these models have also been used *as game engines themselves* (Kim et al., 2020).

There is a strong focus on latent variable evolution (LVE), where algorithms like Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES, Hansen and Ostermeier, 1996), and its MAP-Elites variant (CMA-ME, Fontaine et al., 2020), are used to explore the latent space of the GAN, optimizing towards features like playability, or desired features like floor gaps in platformers (Capps and Schrum, 2021; Edwards, Jiang, and Togelius, 2021; Giacomello, Lanzi, and Loiacono, 2018; Irfan, Zafar, and Hassan, 2019; Schrum,

Volz, and Risi, 2020; Volz et al., 2018). GANs have also been used in systems that adapt and balance content. Rajabi et al. (2021) use a Reinforcement Learning agent to assess the playability and difficulty of levels in the latent space.

VAES VAEs have found several applications in video games. Examples of these include pixel art (Saravanan and Guzdial, 2022), WarCraft 2 and Super Metroid maps (Karth et al., 2021, used alongside Wave Function Collapse), and a plethora of tile-based levels like Super Mario Bros (SMB), Kid Icarus, Megaman, Ninja Gaiden, CastleVania, Lode Runner and Sokoban (Sarkar and Cooper, 2020b; 2021b; Snodgrass and Sarkar, 2020; Zakaria, Fayek, and Hadhoud, 2023).

An area of focus has been the blending of different games by learning and exploring a single latent space for more than one game (Sarkar and Cooper, 2020a; 2021a; b; Sarkar, Yang, and Cooper, 2019).

VAEs can also be used in conjunction with AR models. Tanabe et al. (2021) build stable levels for Angry birds using VAEs with encoders and decoders based on LSTMs.

6.8 DGMS & FUNCTIONAL CONTENT

Unlike images of faces or objects (Dhariwal and Nichol, 2021; Karras, Laine, and Aila, 2019; Karras et al., 2020), video game levels like the ones discussed in our examples need to satisfy a *functionality* criteria (Liu et al., 2020; Summerville et al., 2018). For example, using the VAE trained on SMB levels may result in levels with inaccessible parts due to pipes that are too tall or floor gaps that are too wide. This section discusses the playability of content produced by our VAEs on the SMB and Zelda examples, briefly surveys how this issue is being addressed by the community and finishes with an analogy to other domains like protein modeling and robotics.

PLAYABILITY STRUCTURE IN LATENT SPACE After sampling latent codes at random from the prior $\boldsymbol{z} \sim p(\boldsymbol{z})$, it is fairly common to decode to unplayable levels. Fig. 7.1b shows an example.[14] Similarly, decoded levels of the Zelda model may contain broken doors or lack doors and stairs at all (making them unfeasible) (Figs. 6.8 and 6.9).

In the next chapters, we examine the question of how the playable content is distributed in latent space. To test at scale, we use Robin Baumgarten's A* agent as a proxy for human players. This agent, which won the 2009 Mario AI competition, is usually considered super-human in performance (Togelius, Karakovskiy, and Baumgarten, 2010). This agent is available in

14 Run this experiment yourself! It is available in the code repository with the minimal example of a VAE in SMB: https://github.com/miguelgondu/minimal_VAE_on_Mario#using-the-simulator

the MarioGAN repositories (Volz et al., 2018).[15] Running Baumgarten's A* agent in this framework returns a set of telemetrics, including whether the level was solved, the number of jump action calls, how much of the level was traversed, etc.

To find out how playability is distributed in latent space, we decode a grid akin to the one shown in Fig. 6.7, but with size $G = 50$ instead of $G = 5$. For each one of these evenly-spaced latent codes, we decode the level and test it using Baumgarten's A* agent. Although this simulation is supposed to be deterministic, experiments show that the same level returns different telemetrics in different runs. To account for this stochasticity, we average the telemetrics over 5 runs. The resulting *playability grid* is shown in Fig. 6.10, with blue regions corresponding to playable levels, and white regions corresponding to non-playable. We explore this structure in detail in the next chapters.

Figure 6.10: **Playability in latent space**. A grid of latent codes (Fig. 6.7, with $G = 50$ and the same limits) is decoded and tested using Baumgarten's A* agent. Blue colors correspond to playable regions, and white to non-playable.

RELATED WORK ON PLAYABILITY As this chapter shows, DGMs are widely used in video games. How do researchers and practitioners deal with this playability issue in practice?

Plenty of research is devoted to *level repair*, where the generated levels are repaired post hoc. Jain et al. (2016) use vanilla AEs to fix small chunks of levels in SMB, since the network learns the structure of already playable data and can "fix" by encoding and decoding. They note, however, that this only works for small-sized windows. Zhang et al. (2020) propose a generate-and-repair scheme in which levels for Zelda are fixed post hoc using a *Mixed Integer Linear Programming* optimization. Their key insight is that the problem of repairing a given level can be formalized as a minimal edit problem with constraints (like the uniqueness of the avatar, key and goal, or the path-connectedness of these). Analogously, Gutierrez and Schrum (2020) fix Zelda levels generated by a GAN using an approach based on A*.

Another way to compensate for the unplayable content in latent space is through LVE, discouraging the evolution from picking unplayable levels by assigning them low fitness scores. This is the approach in (Edwards, Jiang, and Togelius, 2021; Irfan, Zafar, and Hassan, 2019; Sarkar and Cooper, 2021b; Schrum, Volz, and Risi, 2020; Volz et al., 2018), among others.

15 https://github.com/CIGbalance/DagstuhlGAN

FUNCTIONAL CONTENT IN OTHER DOMAINS The problem of generating functional content is not unique to video games. Other domains consider the same problem: learning continuous representations of their data using latent variable models and using said models to generate novel samples. Two examples of such domains include protein modeling and robotics.

In protein modeling, discrete VAEs have also been used to learn a latent space of e.g. molecules (Gómez-Bombarelli et al., 2018; Stanton et al., 2022) using their SMILES representation; the problem of decoding to valid molecules has been addressed by proposing alternative representations that *always* decode to functional content (Krenn et al., 2022). The PCGML community could take inspiration from such representations, and find alternative ways to represent game content (as is discussed in (Summerville et al., 2018, Sec. 4.B.)). In robotics, non-Euclidean data is often considered (e.g. quaternions for modeling rotations, see (Jaquier and Asfour, 2022)), and there is a need for respecting this geometric structure when generating trajectories (i.e. decoding to "valid rotations").

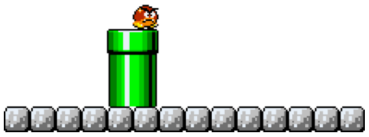
The next chapter takes inspiration from recent works in these two domains (Beik-Mohammadi et al., 2021; Detlefsen, Hauberg, and Boomsma, 2022), and sets the ground for a geometric approach to reliably presenting functional content in tile-based VAEs.

7

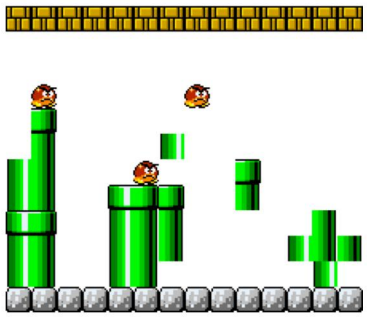
TOWARDS SAFE CONTENT GENERATION USING DIFFERENTIAL GEOMETRY

As we saw in the last chapter, we can use Deep Generative Models to approximate the underlying distribution of a corpus of video game content encoded as a sequence (or grid) of tiles. Unfortunately, we have no guarantees that the content generated by, say, a Variational Autoencoder (VAE) or a Generative Adversarial Network (GAN) will be playable. Fig. 7.1 shows a couple of examples of levels generated from a VAE trained on Super Mario Bros (SMB). In Fig. 7.1b the player cannot jump over the first pipe, while Fig. 7.1a has no impossible obstacles.

This is a well-known problem in learning-based procedural content generation (Summerville et al., 2018, Sec. 4.A), (Liu et al., 2020, Sec. 3). Researchers have addressed this problem by including constraints to probabilistic generative models like Markov chains (Snodgrass and Ontañón, 2016), by fixing the levels post hoc using e.g. pathfinding agents (Cooper and Sarkar, 2020), or by resampling until a playable level is found (Shu, Liu, and Yannakakis, 2021). Alternatives to Deep Generative/probabilistic models have also been proposed, like directly solving a constraint optimization problem at the cost of expensive generation times (Cooper, 2022; Smith and Mateas, 2011). Other approaches for addressing playability (like level repair) are discussed in Sec. 6.8.



(a) Playable



(b) Not playable

Figure 7.1: **Playable (a) and not playable levels (b) from an SMB VAE.**

In this thesis, we propose another approach to the problem of safe content generation focusing on the specific case of generating tile-based content using VAEs. This solution is inspired by recent applications of differential geometry to Deep Generative Models (Arvanitidis, Hansen, and Hauberg, 2018), robotics (Beik-Mohammadi et al., 2021), and protein modeling (Detlefsen, Hauberg, and Boomsma, 2022). At its core, our solution modifies distances in latent space, making it "expensive" for samplers and interpolators to go through regions of unplayable content.

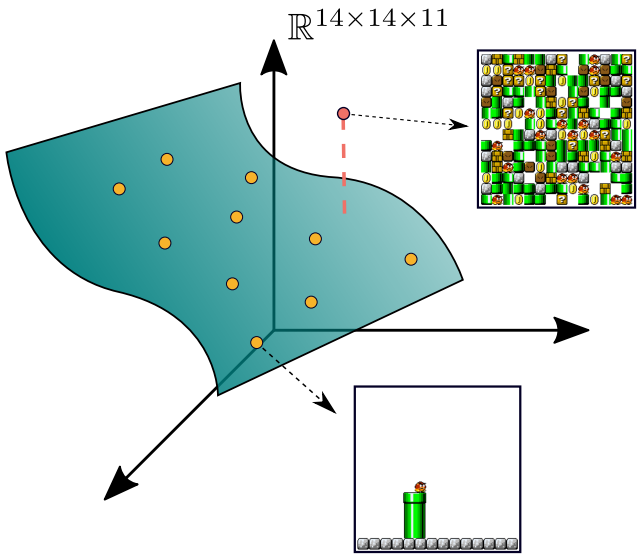


Figure 7.2: **Manifold hypothesis in SMB**. SMB levels live close to a low-dimensional surface on high-dimensional logit space $\mathbb{R}^{14\times14\times11}$. We highlight a level on the surface, which under this hypothesis corresponds to a training example, and a randomly sampled level outside of it. The goal of a representation learning algorithm (like our VAEs) is to approximate this surface.

This chapter starts by introducing differential geometry from an intuitive perspective and continues by discussing related works and recent applications of geometry to DGMs, which serve as a motivation for our geometric approach to generating functional content. We follow by outlining the challenges of applying these ideas. The chapter then ends with an outlook of the rest of this dissertation, explaining how some of these challenges were addressed.

7.1 AN INTUITIVE INTRODUCTION TO DIFFERENTIAL GEOMETRY

Variational Autoencoders (VAEs) are explained in detail in Chap. 6, Sec. 6.4.

To describe the concepts of differential geometry that will be relevant in this thesis, we start by describing our experimental setup.[1] In VAEs (and most other DGMs), a decoder function $\text{dec}_\theta : \mathcal{Z} \to \mathbb{R}^D$ maps low-dimensional data in the latent space $\mathcal{Z} \subseteq \mathbb{R}^d$ to their reconstructions in data space $\mathbb{R}^D$ (where $d < D$). This decoder spans a surface in data space, and differential geometry studies these surfaces and how distances can be defined on them.

Let's make things more tangible by looking at our VAEs trained on the tile-based representation of SMB levels (see Sec. 6.6.1). Recall that this VAE is trained to decode the *logits* $l \in \mathbb{R}^{14\times14\times11}$ of SMB levels, which are then transformed into probability vectors using the Softmax activation. It is hypothesized that all the logits that decode to SMB levels live on a surface,

1 See (Hauberg, 2022) for a more formal introduction to the geometry of generative models. This introduction focuses on our setting: tile-based VAEs trained on video game levels.

and the goal of a VAE (or any other dimensionality reduction technique) is to approximate this surface. To illustrate this, we sample logits uniformly at random in $\mathbb{R}^{14\times14\times11}$ and compare them with decodings of our VAE trained on SMB in Fig. 7.2.

Believing that data lies on a low-dimensional surface is known as the *Manifold Hypothesis* (Bengio, Courville, and Vincent, 2014, Sec. 8). Intuitively speaking, a manifold is a surface that *locally* looks like Euclidean space. This hypothesis is used to justify the fact that dimensionality reduction works: we should be able to describe content with less numbers than the data is originally encoded in. These types of algorithms have been applied to faces (Karras, Laine, and Aila, 2019; Karras et al., 2020), molecules (Gómez-Bombarelli et al., 2018; Stanton et al., 2022), and SMB levels (Sarkar and Cooper, 2020b; Volz et al., 2018), among several other types of content.

In the language of differential geometry, learning latent representations corresponds to learning *charts*: local approximations of the surface that match points on the latent space with points on the surface in a one-to-one fashion (Lee, 2000, Chap. 1). In the case of Variational Autoencoders, the use of approximate posterior distributions makes these *probabilistic charts*, approximating the manifold on average (Fig. 7.3). In other words, the decoder is a probabilistic approximation of the "true" surface that contains all possible SMB levels.

How do we measure distances on this surface? Given our decoder $\text{dec}\colon \mathcal{Z} \to \mathbb{R}^D$,[2] consider a curve $\boldsymbol{c}\colon [0,1] \to \mathcal{Z}$ that starts at $\boldsymbol{z}_0 = \boldsymbol{c}(0)$ and ends at $\boldsymbol{z}_1 = \boldsymbol{c}(1)$. We can measure its length by taking small time steps δt and adding the differences $\|\text{dec}(\boldsymbol{c}(t+\delta t)) - \text{dec}(\boldsymbol{c}(t))\|$. The limit, as we take smaller steps and add them, is the following integral:

$$\begin{aligned}
\text{Length}[\boldsymbol{c}] &= \int_0^1 \left\| \frac{\mathrm{d}}{\mathrm{d}t} \text{dec}(\boldsymbol{c}(t)) \right\| \mathrm{d}t \\
&= \int_0^1 \|\boldsymbol{J}_{\text{dec}}(\boldsymbol{c}(t))\boldsymbol{c}'(t)\| \mathrm{d}t \\
&= \int_0^1 \sqrt{(\boldsymbol{J}_{\text{dec}}(\boldsymbol{c}(t))\boldsymbol{c}'(t))^\top (\boldsymbol{J}_{\text{dec}}(\boldsymbol{c}(t))\boldsymbol{c}'(t))} \mathrm{d}t \\
&= \int_0^1 \sqrt{\boldsymbol{c}'(t)^\top \boldsymbol{J}_{\text{dec}}(\boldsymbol{c}(t))^\top \boldsymbol{J}_{\text{dec}}(\boldsymbol{c}(t))\boldsymbol{c}'(t)} \mathrm{d}t
\end{aligned} \tag{7.1}$$

where $\boldsymbol{J}_{\text{dec}}(\boldsymbol{z})$ is the Jacobian of the decoder, which can be thought of as a first-order derivative. More precisely, this Jacobian measures infinitesimal differences of the decoder:

$$\boldsymbol{J}_{\text{dec}}(\boldsymbol{z}) \approx \left[\frac{\text{dec}(\boldsymbol{z} + \mathrm{d}\boldsymbol{z}_i) - \text{dec}(\boldsymbol{z})}{\|\mathrm{d}\boldsymbol{z}_i\|} \right]_{i=1}^{\dim(\mathcal{Z})}, \tag{7.2}$$

where $\mathrm{d}\boldsymbol{z}_i$ is a small vector in the i-th direction. This approximation is made exact when taking the limit $\|\mathrm{d}\boldsymbol{z}_i\| \to 0$. The matrix

$$\boldsymbol{M}(\boldsymbol{z}) := \boldsymbol{J}_{\text{dec}}(\boldsymbol{z})^\top \boldsymbol{J}_{\text{dec}}(\boldsymbol{z}) \tag{7.3}$$

2 We omit the parameters θ to unclutter the notation.

is known as the **pullback metric** (Lee, 2018, Appendix B.). Replacing in Eq. (7.1), we get

$$\text{Length}[c] = \int_0^1 \sqrt{\boldsymbol{c}'(t)^\top \boldsymbol{M}(\boldsymbol{c}(t))\boldsymbol{c}'(t)}\mathrm{d}t. \tag{7.4}$$

Notice how this equation states that $\boldsymbol{M}(\boldsymbol{z})$ plays a crucial role in measuring lengths on the surface.

Using this definition of the length of a curve, we can further define **geodesics** as curves that locally minimize length (Lee, 2018, Chap. 4). Geodesics are to generic surfaces what straight lines are to Euclidean space. In practice, these geodesics are computed by minimizing a curve's energy instead of length (Arvanitidis et al., 2022; Detlefsen, Hauberg, and Boomsma, 2022).

A benefit of using the distances measured directly on the surface in data space is *invariance to reparametrizations*. Two training runs of the same VAE may produce completely different yet similarly performing decoders. Since their weights are different, the latent embeddings for a pair of points in the first VAE may be very different from that of the second one. However, the geodesic distance between this pair of encodings will remain the same across the two networks since it is measured in data space.

Another important quantity is **metric volume** (Lee, 2018, Chap. 2), given by

$$\text{volume}(\boldsymbol{z}) = m(\boldsymbol{z}) = \sqrt{\det(\boldsymbol{M}(\boldsymbol{z}))}, \tag{7.5}$$

which corresponds to measuring how "big" the surroundings of $z \in \mathcal{Z}$ are. Visually, geodesics will tend to follow the regions of low metric volume.

7.2 MANIPULATING THE GEOMETRY OF LATENT SPACES

The previous section described how lengths and volumes can be measured in latent space. We now discuss how this geometry can be modified to our advantage. This theory was developed for *Gaussian* VAEs in (Arvanitidis, Hansen, and Hauberg, 2018) and then adapted to spherical (Beik-Mohammadi et al., 2021) and Categorical data (Detlefsen, Hauberg, and Boomsma, 2022).

Circling back to the probability theory involved in VAEs, remember that they learn two distributions: an approximation of the posterior over the latent codes $q_phi(\boldsymbol{z}|\boldsymbol{x})$, which is usually a Normal distribution with mean and covariance parametrized by the encoder, and a likelihood distribution $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ which, depending on the nature of the data, can be a Normal if the data is continuous, a Categorical if the data is discrete, etc.

7.2.1 *Defining latent space geometries for Gaussian decoders*

Let's start by focusing on the case in which $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is a Normal distribution, say $p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{z}), \boldsymbol{\sigma}_\theta(\boldsymbol{z})^2)$. The decoder takes the following form:

This is known as the reparametrization trick. We explain it in detail in Sec. 6.4.4.

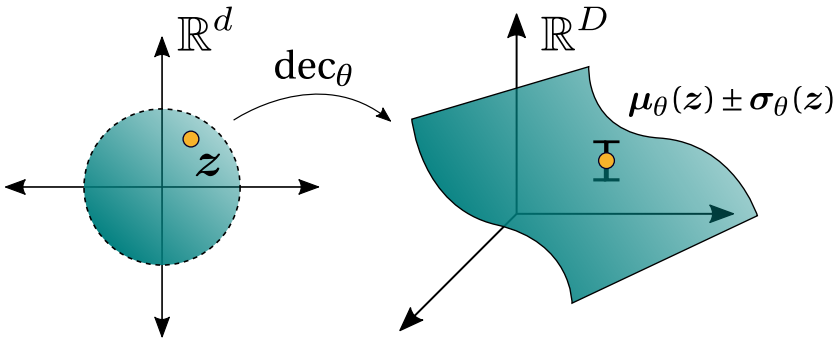


Figure 7.3: **Gaussian VAEs as probabilistic charts**. Gaussian VAEs learn a mean surface and estimate the uncertainty around it. Here, we show our latent space $\mathcal{Z} \subseteq \mathbb{R}^d$ and its image under the decoder. We map a point $\boldsymbol{z}$ to $\boldsymbol{\mu}_\theta(\boldsymbol{z})$, with uncertainty $\boldsymbol{\sigma}_\theta(\boldsymbol{z})$.

$$\text{dec}_\theta(\boldsymbol{z}) = \boldsymbol{\mu}_\theta(\boldsymbol{z}) + \varepsilon \odot \boldsymbol{\sigma}_\theta(\boldsymbol{z})^2, \;\; \boldsymbol{\mu}_\theta, \boldsymbol{\sigma}_\theta : \mathbb{R}^d \to \mathbb{R}^D, \; \varepsilon \sim N(\mathbf{0}, \boldsymbol{I}_D). \tag{7.6}$$

As discussed above, the image of this decoder in data space $\mathbb{R}^D$ is a *random surface*: a mean surface $\boldsymbol{\mu}_\theta(\mathcal{Z})$ with uncertainties around it (see Fig. 7.3).

To compute distances and volumes, we need to compute the pullback metric $\boldsymbol{M}(\boldsymbol{z}) = \boldsymbol{J}_{\text{dec}}(\boldsymbol{z})^\top \boldsymbol{J}_{\text{dec}}(\boldsymbol{z})$ first. Since Eq. (7.6) has a Gaussian random variable in it, $\boldsymbol{M}(\boldsymbol{z})$ itself becomes a random variable. In other words, we need to discuss the value that the metric $\boldsymbol{M}(\boldsymbol{z})$ takes *on average*.[3]

Arvanitidis, Hansen, and Hauberg proved that

$$\mathbb{E}_{\varepsilon \sim N(\mathbf{0}, \boldsymbol{I}_D)}\left[\boldsymbol{M}(\boldsymbol{z})\right] = \boldsymbol{J}_{\boldsymbol{\mu}_\theta}(\boldsymbol{z})^\top \boldsymbol{J}_{\boldsymbol{\mu}_\theta}(\boldsymbol{z}) + \boldsymbol{J}_{\boldsymbol{\sigma}_\theta}(\boldsymbol{z})^\top \boldsymbol{J}_{\boldsymbol{\sigma}_\theta}(\boldsymbol{z}). \tag{7.7}$$

Notice that this expression for the pullback contains the Jacobian of the standard deviation $\boldsymbol{\sigma}_\theta(\boldsymbol{z})$. The geometry of the latent space is intimately linked to how good our uncertainty quantification is (i.e. whether $\boldsymbol{\sigma}_\theta(\boldsymbol{z})$ is high in the parts of the latent space we want to avoid).

VAEs are notorious for their lack of good uncertainty quantification (Rybkin, Daniilidis, and Levine, 2021). Fig. 6.1 shows the value of $\boldsymbol{\sigma}_\theta$ for a fine grid in latent space in our MNIST(1) VAE, which was trained on only the digit "1" (see Sec. 6.4.6). Ideally, we would see that our network has low variance around the training codes, and is uncertain in the "unknown" regions of latent space.

Several attempts to fix this lack of proper uncertainty quantification have been proposed. Some are principled, and modify the training of the networks to better estimate the uncertainty (Rybkin, Daniilidis, and Levine, 2021; Skafte, Jørgensen, and Hauberg, 2019). However, the community has found it more practical to rely on post hoc fixes.

Originally, Arvanitidis, Hansen, and Hauberg (2018) proposed training a Radial Basis Function (RBF) network after training the VAE, encouraging

3 This treatment of the Jacobian as a random variable is **not** completely formal since it is only defined for a fixed ε in Eq. (7.6). For a formal treatment, we recommend (Eklund and Hauberg, 2019), where the authors treat this Gaussian decoder as a random projection of the function $h(\boldsymbol{z}) = [\boldsymbol{\mu}(\boldsymbol{z}), \boldsymbol{\sigma}(\boldsymbol{z})]$, given by concatenating the mean and standard deviation.

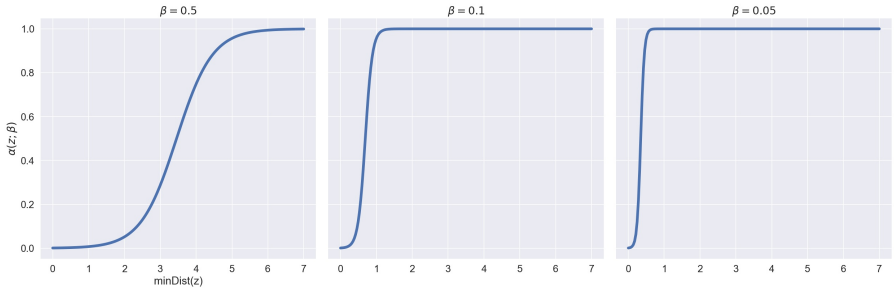


Figure 7.4: **Translated sigmoids** $\alpha(\boldsymbol{z};\beta)$ as a function of the distance to the centers minDist($\boldsymbol{z}$). For lower values of the hyperparameter $\beta > 0$ we get a translated sigmoid that raises to 1 faster.

the RBF to assign low variance close to the latent codes of the training set. A year later Skafte, Jørgensen, and Hauberg proposed a heuristic for training these variance networks which relies on a "translated sigmoid", which works as a semaphore (Skafte, Jørgensen, and Hauberg, 2019, Sec. 3.5). This proposed heuristic has stuck and was used in e.g. (Arvanitidis et al., 2022; Detlefsen, Hauberg, and Boomsma, 2022). Since we also use it, we will discuss it in detail in what remains of this section.

7.2.2 *Proper uncertainty quantification using a translated sigmoid*

To recap: the geometry of the latent space of a VAE is intimately linked to the variance output $\boldsymbol{\sigma}_\theta(\boldsymbol{z})$, but training VAEs the usual way results in poor estimates of this uncertainty. Thus, we fix the estimate of the variance post hoc.

Putting it more formally, we want to replace the standard deviation $\boldsymbol{\sigma}_\theta(\boldsymbol{z})$ that is originally learned by the network with a different one (denoted $\widetilde{\boldsymbol{\sigma}}_\theta(\boldsymbol{z})$) which behaves like this:

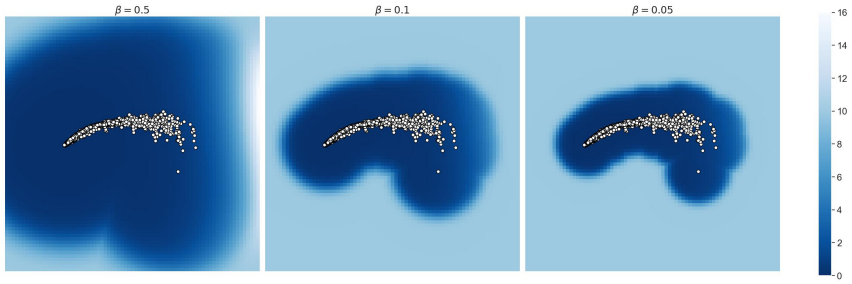
$$\widetilde{\boldsymbol{\sigma}}_\theta(\boldsymbol{z}) = \begin{cases} \boldsymbol{\sigma}_\theta(\boldsymbol{z}) & \text{if } z \text{ is close to the training codes,} \\ \text{a large number} & \text{otherwise.} \end{cases} \tag{7.8}$$

We achieve this by considering the extrapolation heuristic proposed in (Skafte, Jørgensen, and Hauberg, 2019): first, train K-means on the latent codes of the training set, arriving at K different centers $\mathscr{C} = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K\}$. Let minDist($\boldsymbol{z}$; $\mathscr{C}$) be the minimum of the distances $\{\text{dist}(\boldsymbol{z}, \boldsymbol{c}_k)\}_{k=1}^K$ as a function of $\boldsymbol{z}$. With this, define

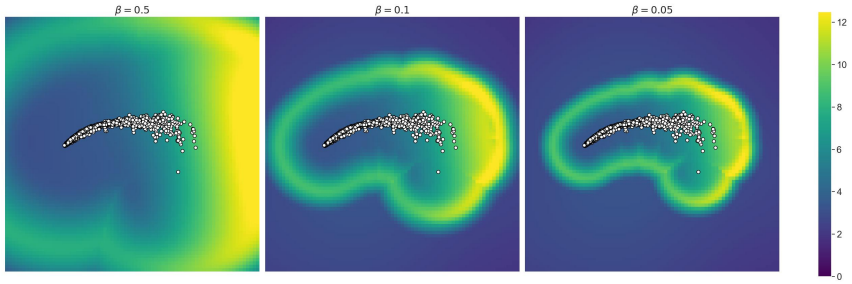
$$\alpha(\boldsymbol{z};\, \beta, \mathscr{C}) = \text{Sigmoid}\left(\frac{\text{minDist}(\boldsymbol{z};\, \mathscr{C}) - \beta s}{\beta}\right) \tag{7.9}$$

where $\beta > 0$ is a hyperparameter and $s \approx 6.9$.[4] $\alpha(\boldsymbol{z};\beta)$ is approximately 0 when $\boldsymbol{z}$ is close to the training centers $\boldsymbol{c}_k$, and approximately 1 when far away. The slope by which this function moves from 0 to 1 is governed by β, the impact of which is shown in Fig. 7.4.

4 This value is chosen by solving for $\alpha(0) \approx 0$. To be exact, we used $s = 6.9077542789816375$.



(a) Extrapolation of uncertainty $\boldsymbol{\sigma}_\theta(\boldsymbol{z})$.



(b) Log-metric volume in latent space.

Figure 7.5: **Impact of the hyperparameter** β. As shown in Fig. 7.4, the hyperparameter β governs how "quickly" the VAE extrapolates to uncertainty in the mechanism proposed by Skafte, Jørgensen, and Hauberg. This figure shows the modified decoder's uncertainty above and the induced metric volume below. Lower values of β allow for quicker extrapolation, which induces a "wall" of metric volume.

Using this "translated sigmoid", we define $\widetilde{\boldsymbol{\sigma}}_\theta(\boldsymbol{z})$ by

$$\widetilde{\boldsymbol{\sigma}}_\theta(\boldsymbol{z};\beta) = (1-\alpha(\boldsymbol{z};\beta))\boldsymbol{\sigma}_\theta(\boldsymbol{z}) + 10\alpha(\boldsymbol{z};\beta). \tag{7.10}$$

If $\boldsymbol{z}$ is close to the training codes, $\alpha(\boldsymbol{z}) \approx 0$ and thus our standard deviation returns what the network learned; on the other hand, if $\boldsymbol{z}$ is far away, $\alpha(\boldsymbol{z}) \approx 1$ and the network decodes to 10 (which is chosen as a token for high variance). This fulfills our desiderata in Eq. (7.8). Fig. 7.5a shows how the standard deviation gets corrected for our MNIST(1) example, and the impact on the metric volume is illustrated in Fig. 7.5b.[5]

Originally, Skafte, Jørgensen, and Hauberg initialize β and the centers $\{\boldsymbol{c}_k\}_{k=1}^K$ as described above, and optimize them during training. We found the optimization unnecessary in practice, and settle for leaving them static.

7.2.3 *Defining latent space geometries for Categorical decoders*

The extrapolation mechanism described in the previous subsection assumes that the decoder outputs the parameters of a Gaussian distribution. How does this extrapolation mechanism work for the Categorical?

The reader might find it helpful to recall VAEs with Categorical decoders (see Sec. 6.5).

5 The code used for generating these Figs. is available in `https://github.com/miguelgondu/examples_in_thesis`.

Let's start with a small recap of how Categorical VAEs work: the decoder learns a discrete probability distribution over a set of tiles, or tokens, denoted by $\{t_1, \dots, t_C\}$. More explicitly, a decoder with L outputs $\text{dec}_\theta(\boldsymbol{z}) = [\boldsymbol{p}_1, \dots, \boldsymbol{p}_L]$ learns a probability vector $\boldsymbol{p}_l = [p_{l,1}, \dots, p_{l,C}]$ for each position $l \in \{1, \dots, L\}$, estimating the probabilities that the tile in position l is t_c for each $c \in \{1, \dots, C\}$. In symbols:

$$\text{dec}(\boldsymbol{z})_{l,c} = \text{Prob}\left[\text{tile } x_l = t_c\right].$$

Detlefsen, Hauberg, and Boomsma (2022) modify the extrapolation mechanism described for the Gaussian by maximizing the *entropy* of the Categorical distribution. In this scenario, maximal variance is achieved when the probability of each one of the classes is $1/C$. With this notation, the extrapolation modifies the decoder as follows

$$\widetilde{\text{dec}}(\boldsymbol{z})_{l,c} = \alpha(\boldsymbol{z}; \beta)\, \text{dec}(\boldsymbol{z})_{l,c} + (1 - \alpha(\boldsymbol{z}; \beta))(1/C). \tag{7.11}$$

Putting it another way: the modified extrapolation mechanism decodes to the probabilities learned during training when close to the training codes, and decodes to a probability vector that is at random as possible when $\boldsymbol{z}$ is far from the training codes.

7.3 AN APPLICATION IN ROBOTICS

Robotic data naturally lives on geometrically constrained spaces (Jaquier and Asfour, 2022; Jaquier et al., 2021). For example, the positions of a robot arm can be parametrized by their 3D location and the orientation of the gripper (the combination of which is denoted by $\mathbb{R}^3 \times \mathbb{S}^3$, where $\mathbb{S}^3$ are the quaternions), or by its joint angles (each of which lives in a copy of the circle, denoted $\mathbb{S}^1$). When learning a latent space of such positions or joint angles, we would need a distribution that decodes to quaternions or the circle respectively.

In 2021, Beik-Mohammadi et al. used VAEs that decode to the von Mises-Fisher (vMF) distribution on $\mathbb{S}^3$ to learn from demonstrations of a robot arm with 7 degrees of freedom. Since the vMF distribution also allows for reparametrization, they derive an expression for the metric volume that is similar to the one described in Eq. (7.5).

With this latent geometry, they are able to synthesize robot motion safely even across multiple demonstrations. This is achieved by following geodesic paths in latent space since these stay within the demonstrations provided in the training set.

Their application is close in nature to ours: there are regions in latent space they want to stay within when interpolating between two points, so they modify the geometry of the latent space to make it "expensive" for the interpolations to go out of the support of the data. In our method, we want to stay close to playable regions of the latent space when sampling and interpolating.

One lesson from this application is: when dealing with novel types of data and distributions (like quaternions and the vMF), the metric needs to be derived again by computing the respective Jacobians. We consider this to be a drawback of such an approach.

7.4 AN APPLICATION IN PROTEIN MODELING

This second application deals with data that is closer in nature to tile-based video game levels. Proteins and molecules, like tile-based levels, are modeled as sequences of tokens (e.g. as "strings" of amino acids or RNA) (Stanton et al., 2022, Sec. 2.1.).

As is discussed in Sec. 7.2, proper uncertainty quantification is necessary for the latent space geometry to be well-behaved. The extrapolation mechanism for Categorical data described earlier was first proposed by Detlefsen, Hauberg, and Boomsma.

Another issue arises: computing the metric $\boldsymbol{M}(\boldsymbol{z})$ for a Categorical distribution is not as trivial as in the Gaussian case: the data does not live in continuous, Euclidean space but rather in sequence space. The authors compute a curve's energy by measuring the expected squared distance between the one-hot encodings of the decoded sequences. In short, dealing with Categorical data forces us to consider different distances.

Detlefsen, Hauberg, and Boomsma study the VAE latent space of aligned sequences of amino acids for a specific family of proteins (beta-lactamase), and find that geometry allows for interpretability. Their latent space resembles a hierarchy, with geodesics generally following a phylogenetic tree estimated from the protein sequences in the training set.

7.5 APPLYING GEOMETRY TO VIDEO GAME CONTENT: CHALLENGES

These three previous sections outline several challenges of learning latent geometries:

1. Depending on the nature of the data, we need to either modify the notion of distance or compute the expected metric.
2. Computing expected metrics involves computing Jacobians, which is involved even when using current autodifferentiation software like `PyTorch` (Paszke et al., 2019).[6]
3. Given a distribution, we need to specify an extrapolation mechanism. Extrapolating to uncertainty in the Gaussian is different to extrapolating to uncertainty in the Categorical.

6 JAX is a strong competitor in this front. Its approach to automatic differentiation has the computation of the Jacobian easily exposed. https://jax.readthedocs.io/en/latest/notebooks/autodiff_cookbook.html#jacobians-and-hessians-using-jacfwd-and-jacrev

4. Something that might not be apparent at first is the sensitivity to hyperparameters. We glossed over hyperparameters like the "steepness" of the translated sigmoid β or the number of centers K, but these turn out to be *essential* for geodesics and random walks to be well-behaved. Tuning is currently done by hand, using trial-and-error and relying on qualitative assessments, like how the latent space *looks*.
5. Since we rely on visual inspection, these methods have mostly been explored in latent spaces of dimension 2. Scaling them to higher dimensions reliably would require finding more robust ways of tuning the hyperparameters involved.
6. As can be seen in Fig. 7.5b, pulling back the metric from data space usually translates to building a "wall" of metric volume around the support of the data. Outside this wall, points are close-by. If safety is the main concern, we would like *all* regions outside the region of interest to be expensive to get to.

In the particular case of video game content, items 1 and 6 are especially relevant: since our data is Categorical, a different notion of distance needs to be computed (like Detlefsen, Hauberg, and Boomsma), and we would like *all* unplayable regions to be unreachable. Item 2 is being addressed by developing frameworks on top of `PyTorch`, which give easy access to Jacobians (Detlefsen et al., 2021, `stochman`).

7.6 CONCLUSION & OUTLOOK

This chapter introduced differential geometry as a potential tool to solve the problem of reliably generating functional content in latent space, exemplified other applications of geometry in DGMs, and outlined several challenges to overcome in order to apply this theory to video game content.

In the following chapter, item 1 is addressed by defining latent space geometries for (almost) any distribution using tools from Information Geometry (IG). Chap. 9 covers a method for learning latent space geometries in VAEs that decode to tile-based content safely, addressing item 6 and tackling the problem of reliably generating playable content.

8

DEFINING LATENT SPACE GEOMETRIES FOR (ALMOST) ANY DISTRIBUTION

This chapter presents a new methodology for learning latent space geometries, one that allows for decoding to (almost) any distribution. This addresses one of the issues we raised about learning latent space geometries in the previous chapter: the fact that, for each new distribution, the expected pullback metric $\boldsymbol{M}(\boldsymbol{z})$ (see Eq. (7.3)) needs to be re-computed.

Learning latent space geometries in this generalized setting can be achieved by using the tools of *Information Geometry* (IG) (Nielsen, 2020; 2022). Information geometers have been studying probability distributions from a Riemannian lense, and they have introduced the notion of a *statistical manifold* $\mathcal{H}$, the manifold of the parameters of a given distribution, coupled with the *Fisher-Rao metric* (or the *Fisher information matrix*).

At its core, this proposed method re-frames the decoder of Variational Autoencoders (VAEs): instead of decoding to data space, the decoder returns the parameters of the distribution in a statistical manifold, from which the Fisher-Rao metric can be pulled back. In other words, our previous approaches pulled back the *Euclidean* metric from data space $\mathbb{R}^D$, and now we focus on pulling back the *Fisher-Rao* metric from $\mathcal{H}$.

The topics this chapter discusses are theoretical and not exclusively related to video games. Readers that are not interested in latent space geometries could skip this chapter without any cost. Still, the math-heavy sections of this chapter are summarized by giving a short, intuitive description of the theoretical discussion.

This chapter starts with a formalization of the intuitive presentation of differential geometry presented in earlier chapters, followed by the presentation of our novel methodology, including two experiments, and finishes with a practical implementation in Python using tools like `PyTorch` (Paszke et al., 2019) and `stochman` (Detlefsen et al., 2021). The contents of this chapter are based on the contribution (Arvanitidis, González-Duque, Pouplin, Kalatzis, and Hauberg, 2022).

8.1 REVISITING DIFFERENTIAL GEOMETRY

In Sec. 7.1 the notion of the *pullback metric* $\boldsymbol{M}(\boldsymbol{z}) = \boldsymbol{J}_{\text{dec}}(\boldsymbol{z})^\top \boldsymbol{J}_{\text{dec}}(\boldsymbol{z})$ was introduced. To re-cap, if $\boldsymbol{c}\colon [0,1] \to \mathcal{Z}$ is a curve in latent space and $\text{dec}\colon \mathcal{Z} \to \mathbb{R}^D$ is a decoder, the length of the curve in latent space is measured using distances in ambient space (Eq. (7.4)):

$$\text{Length}[\boldsymbol{c}] = \int_0^1 \sqrt{\boldsymbol{c}'(t)^\top \boldsymbol{M}(\boldsymbol{c}(t))\boldsymbol{c}'(t)}\,\mathrm{d}t.$$

Notice how, to define the length of a curve in latent space, we need to compute an inner product *with its derivatives*. Geometrically speaking, these derivatives are vectors that are tangent to the curves. These two facts motivate the following three mathematical definitions, which formalize what was previously discussed:

TANGENT SPACE Given a manifold $\mathcal{M}$, its tangent space at a point $p \in \mathcal{M}$ (denoted $T_p\mathcal{M}$) consists of all the derivatives of curves that pass through p. We denote (and think of) such derivatives as vectors. Indeed, one sometimes considers points on the manifold in a coordinate system, and this abstract notion of "vectors as derivatives" turns computational, with $T_p\mathcal{M}$ being easily identifiable with a real vector space (Lee, 2000).

METRIC Secondly, a *metric* g on $\mathcal{M}$ takes two tangent vectors $\boldsymbol{v}_p, \boldsymbol{w}_p$ at p and returns a real number $g(\boldsymbol{v}_p, \boldsymbol{w}_p)$ such that

- it is symmetric, i.e. $g_p(\boldsymbol{v}_p, \boldsymbol{w}_p) = g_p(\boldsymbol{w}_p, \boldsymbol{v}_p)$.
- it is positive definite, i.e. $g_p(\boldsymbol{v}_p, \boldsymbol{v}_p) \geq 0$, with equality only when $\boldsymbol{v}_p$ is the zero vector.

A manifold M equipped with a metric is known as a *Riemannian manifold*. Taking coordinates, metrics are symmetric positive-definite matrices. An example Riemannian manifold is precisely the Euclidean space $\mathbb{R}^D$ equipped with the identity matrix $\boldsymbol{I}$ as the metric in all tangent spaces.

PULLBACK METRIC Finally, the definition of a pullback metric provided in Eq. 7.3 can be generalized to Riemannian manifolds. If $f : \mathcal{Z} \to (\mathcal{M}, g)$ is a smooth map, a metric on $\mathcal{Z}$ can be defined by computing the inner product on $\mathcal{M}$ and "bringing it back". This pullback metric, denoted by $(f^* g)$, is given by

$$(f^* g)_{\boldsymbol{z}}(\boldsymbol{v}_{\boldsymbol{z}}, \boldsymbol{w}_{\boldsymbol{z}}) = g_{f(\boldsymbol{z})}((\mathrm{d}f)_{\boldsymbol{z}}(\boldsymbol{v}_{\boldsymbol{z}}), (\mathrm{d}f)_{\boldsymbol{z}}(\boldsymbol{w}_{\boldsymbol{z}})), \tag{8.1}$$

where $(\mathrm{d}f)_{\boldsymbol{z}}$ is the differential, which maps tangent vectors $\boldsymbol{v}_{\boldsymbol{z}} \in T_{\boldsymbol{z}}\mathcal{Z}$ to tangent vectors at $T_{f(\boldsymbol{z})}M$. In coordinates, this differential takes the form of the usual Jacobian matrix described in Eq. (7.2).

> SUMMARY The intuitive introduction to differential geometry presented in Sec. 7.1 is formalized by introducing the mathematical definitions of the tangent spaces, metrics, Riemannian manifolds, and pullback metrics.

8.2 DATA SPACE VS. PARAMETER SPACE

Using this language about pullback metrics, the type of pullback metric discussed in Chap. 7 is described formally. Namely, we were *pulling back*

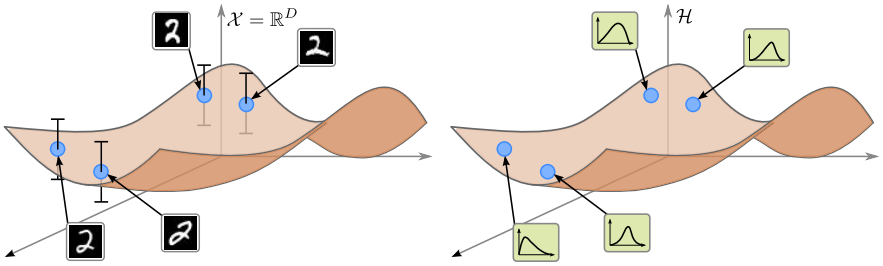


Figure 8.1: **From data space to parameter space**. Instead of using the data space (left) to define distances in latent space, using the parameter space (right) allows for defining latent space geometries to latent spaces of VAEs that decode to (almost) any distribution.

the Euclidean metric from data space $\mathbb{R}^D$. This pullback metric would take two tangent vectors $\boldsymbol{v}_{\boldsymbol{z}}, \boldsymbol{w}_{\boldsymbol{z}} \in T_{\boldsymbol{z}}\mathcal{Z}$ in the latent space and output

$$\begin{aligned}(\text{dec}^*\, \boldsymbol{I})_{\boldsymbol{z}}(\boldsymbol{v}_{\boldsymbol{z}}, \boldsymbol{w}_{\boldsymbol{z}}) &= \boldsymbol{I}_{\text{dec}(\boldsymbol{z})}((\text{d}\,\text{dec})_{\boldsymbol{z}}(\boldsymbol{v}_{\boldsymbol{z}}), (\text{d}\,\text{dec})_{\boldsymbol{z}}(\boldsymbol{w}_{\boldsymbol{z}})) \\ &= (\boldsymbol{J}_{\text{dec}}(\boldsymbol{z})\boldsymbol{v}_{\boldsymbol{z}})^\top \boldsymbol{I}_{\text{dec}(\boldsymbol{z})} \boldsymbol{J}_{\text{dec}}(\boldsymbol{z})\boldsymbol{w}_{\boldsymbol{z}}\end{aligned}$$

but the Euclidean metric $\boldsymbol{I}_{\text{dec}(\boldsymbol{z})}$ is always equal to the identity matrix, which means that

$$(\text{dec}^*\, \boldsymbol{I})_{\boldsymbol{z}}(\boldsymbol{v}_{\boldsymbol{z}}, \boldsymbol{w}_{\boldsymbol{z}}) = \boldsymbol{v}_{\boldsymbol{z}}^\top \boldsymbol{J}_{\text{dec}}(\boldsymbol{z})^\top \boldsymbol{J}_{\text{dec}}(\boldsymbol{z})\boldsymbol{w}_{\boldsymbol{z}},$$

and this implies that the pullback metric $(\text{dec}^*\, \boldsymbol{I})_{\boldsymbol{z}}$ is precisely the usual product of Jacobians discussed in Eq. (7.3).

However, we could also think of the decoder as outputting values in *parameter space* instead, with e.g. $\text{dec}(\boldsymbol{z}) = (\boldsymbol{\mu}(\boldsymbol{z}), \boldsymbol{\sigma}(\boldsymbol{z}))$ in the Gaussian case. For any distribution $p(\boldsymbol{x}|\eta)$ (for example $N(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\sigma})$, or $\text{Cat}(\boldsymbol{x}|\text{probs})$), its *parameter space* (or *statistical manifold*) is defined as the Riemannian manifold given by its set of parameters $\mathcal{H}$, and the *Fisher-Rao metric* (Nielsen, 2022) defined by

$$\boldsymbol{I}_{\text{FR}}(\eta) = \int_{\mathcal{X}} [\nabla_\eta \log p(\boldsymbol{x}|\eta) \nabla_\eta \log p(\boldsymbol{x}|\eta)^\top] p(\boldsymbol{x}|\eta) \text{d}\boldsymbol{x}. \tag{8.2}$$

This change in perspective is illustrated in Fig. 8.1.

The metric $\boldsymbol{I}_{\text{FR}}$ measures the covariance of the *score function* $\nabla_\eta \log p(\boldsymbol{x}|\eta)$. It has seen plenty of use in improving optimization schemes (Amari, 1998; Martens, 2020), as well as training so-called *natural* strategies in RL and evolution (Wierstra et al., 2014).

The main reason for choosing $\boldsymbol{I}_{\text{FR}}$ as a reference metric to pull back is that distances can be computed in parameter space using the *Kullback-Leibler divergence* (KL) locally around a given point $\boldsymbol{z} \in \mathcal{Z}$. The next section explains this in detail.

SUMMARY Instead of using the data space to define distances, we use the space of parameters of the distribution the VAE decodes to. This space has a principled way of measuring distances (given by the Fisher information matrix).

8.3 PULLING BACK THE FISHER-RAO

Let's start by re-introducing the KL divergence, which was presented in the context of training VAEs in Sec. 6.4.3. The KL divergence (Eq. (6.9)) measures how *similar* two distributions $p(\boldsymbol{x}|\eta_1)$ and $q(\boldsymbol{x}|\eta_2)$ are, more precisely:

$$\begin{aligned}\mathrm{KL}(p\,||\,q) &= \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x}|\eta_1)}[\log(p(\boldsymbol{x}|\eta_1)/q(\boldsymbol{x}|\eta_2))] \\ &= \int_{\mathscr{X}} (\log(p(\boldsymbol{x}|\eta_1)) - \log(q(\boldsymbol{x}|\eta_2)))p(\boldsymbol{x}|\eta_1)\mathrm{d}\boldsymbol{x}. \end{aligned} \tag{8.3}$$

These expected values have closed form for most distributions, and they already come implemented in scientific programming packages for probability distributions like `torch.distributions`.

Divergences behave almost like distances, with e.g. the KL divergence between a distribution p and itself being 0. However, unlike distances, divergences are not symmetric and they do not satisfy the triangle inequality.

The KL divergence and the Fisher-Rao metric are closely linked. It is well-known that measuring distances with the Fisher-Rao metric amounts to computing KL divergences locally. Given a curve $\boldsymbol{c}\colon [0,1] \to \mathscr{H}$ in parameter space, we measure its length as usual

$$\mathrm{Length}[\boldsymbol{c}] = \int_0^1 \sqrt{\boldsymbol{c}'(t)^\top \boldsymbol{I}_{\mathrm{FR}}(\boldsymbol{c}(t))\boldsymbol{c}'(t)}\mathrm{d}t.$$

This inner product inside the square root is, locally, just an evaluation of the KL divergence: let $\delta t > 0$, then it can be shown (Nielsen, 2022, pg. 37) that

$$\boldsymbol{c}'(t)^\top \boldsymbol{I}_{\mathrm{FR}}(\boldsymbol{c}(t))\boldsymbol{c}'(t) = \frac{2}{\delta t^2}\mathrm{KL}(p(\boldsymbol{x}|\boldsymbol{c}(t)), p(\boldsymbol{x}|c(t+\delta t))) + o(\delta t^2). \tag{8.4}$$

Replacing this in the equation for the length and taking δt to be small enough (so as to make the $o(\delta t^2)$ term disappear) we have

$$\mathrm{Length}[\boldsymbol{c}] \approx \frac{\sqrt{2}}{\delta t}\int_0^1 \sqrt{\mathrm{KL}(p(\boldsymbol{x}|\boldsymbol{c}(t)), p(\boldsymbol{x}|\boldsymbol{c}(t+\delta t)))}\mathrm{d}t. \tag{8.5}$$

Dividing this curve into T segments of width δt such that $\delta t = 1/T$, this integral can be further approximated by adding up the areas of the rectangles of shape δt times the integrand:

$$\begin{aligned}\text{Length}[\boldsymbol{c}] &\approx \frac{\sqrt{2}}{\delta t}\int_0^1 \sqrt{\text{KL}(p(\boldsymbol{x}\,|\,\boldsymbol{c}(t)), p(\boldsymbol{x}\,|\,\boldsymbol{c}(t+\delta t)))}\,\mathrm{d}t \\ &\approx \frac{\sqrt{2}}{\delta t}\sum_{k=0}^{T-1} \delta t \sqrt{\text{KL}(p(\boldsymbol{x}\,|\,\boldsymbol{c}(k\delta t)), p(\boldsymbol{x}\,|\,\boldsymbol{c}((k+1)\delta t)))} \\ &= \sum_{k=0}^{T-1} \sqrt{2\,\text{KL}(p(\boldsymbol{x}\,|\,\boldsymbol{c}(k\delta t)), p(\boldsymbol{x}\,|\,\boldsymbol{c}((k+1)\delta t)))}. \end{aligned} \tag{8.6}$$

This gives us a recipe for minimizing the length of a given curve $\boldsymbol{c}$: divide it into small subsegments of small width δt, and compute the KL divergence of the decoded distributions. A similar argument can be used to prove that the energy of a curve can be approximated by

$$\text{Energy}[\boldsymbol{c}] \approx \frac{2}{\delta t}\sum_{k=0}^{T-1} \text{KL}(p(\boldsymbol{x}\,|\,\boldsymbol{c}(k\delta t)), p(\boldsymbol{x}\,|\,\boldsymbol{c}((k+1)\delta t))), \tag{8.7}$$

which is better for optimization schemes, since the divergence term is not inside of a square root.

In practice, we use the toolset for parametrizing curves using cubic splines discussed in Sec. 7.1 (Detlefsen et al., 2021, `stochman`). The approximated length and energy in Eqs. (8.6) and (8.7) can be minimized with respect to the parameters of e.g. a cubic spline as long as the distributions allow for computing the KL divergence with autodifferentiation. An example implementation is discussed later in this chapter (Sec. 8.6).

SUMMARY Measuring the lengths of curves in the space of parameters amounts to computing the *Kullback-Leibler* divergence in a fine discretization of the curve. Finding the curves that minimize this length is achieved by minimizing an approximation of the energy, which is differentiable for most distributions VAEs decode to.

8.4 EXPERIMENT: DECODING TO SEVERAL DISTRIBUTIONS

We start by considering a simple set-up from a hand-crafted latent space to the parameter space of several distributions. This hand-crafted latent space is composed of noisy circular data, built by randomly sampling angles and placing points on said angles, with random offsets from the unit circle.

Using this toy latent space, we constructed decoders that map to several parameter spaces: for the Normal, Bernoulli, Beta, Dirichlet, and Exponential distributions. These decoders f_{dist} were randomly initialized neural networks from $\mathcal{Z}_{\text{toy}} = \mathbb{R}^2$ to their parameters.[1]

1 See Sec. A.3.4 in the appendix for all details. The implementation of these experiments is also available in: https://github.com/MachineLearningLifeScience/stochman/tree/black-box-random-geometry/examples/black_box_random_geometries

To manipulate these latent space geometries to prefer the support of the data, the following extrapolations are used:

- For the Normal, we extrapolate to high values of $\sigma(\boldsymbol{z})$.
- For the Bernoulli, the "most uncertain" version of the distribution is given by $\text{probs}(\boldsymbol{z}) = 1/2$.
- For the Beta, the parameters $(\alpha(\boldsymbol{z}), \beta(\boldsymbol{z})) = (1, 1)$ specify a flat, uniform distribution.
- For the Dirichlet, whose samples are probability vectors, $\alpha(\boldsymbol{z}) = 1$ for all classes specifies a uniform distribution over them.
- For the Exponential, small values of the parameter (i.e. $\lambda(\boldsymbol{z}) \to 0$) increase the distribution's variance.

Fig. 8.2 shows several geodesics in latent space, these were cubic splines whose parameters were optimized by minimizing the energy in Eq. (8.7). Pulling back the Fisher-Rao through the modified decoder allows for interpolations that stay within the support of the data.

One problematic latent geometry comes from pulling back the metric for the Bernoulli distribution, which we highlight in Fig. 8.2. Notice how some of the geodesics converge to curves outside of the support. We hypothesize that extrapolating to $\text{probs}(\boldsymbol{z}) = 1/2$ does not provide a "steep" enough difference. Distributions like the Gaussian allow for an abrupt change, since $\sigma(\boldsymbol{z})$ can be chosen to be as high as desired, but this *unboundedness* of the parameters does not happen for the Bernoulli nor the Categorical. The problem of defining reliable latent space geometries for discrete decoders is addressed by introducing a hierarchical layer in the decoder in Chap. 9.

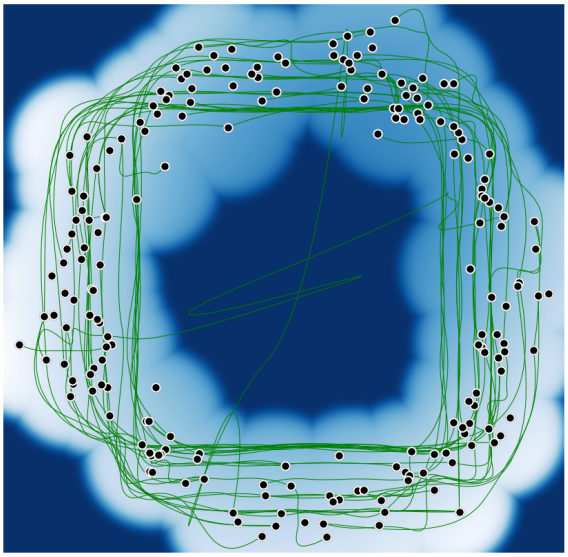
8.5 EXPERIMENT: MODELLING HUMAN POSES

Poses are encoded in terms of the position and orientation of limbs, data that naturally lives on the sphere. A human pose, then, can be seen as a point in a product of spheres (Tournier et al., 2009), data that can be modeled using a product of von Mises-Fisher distributions (vMFs).

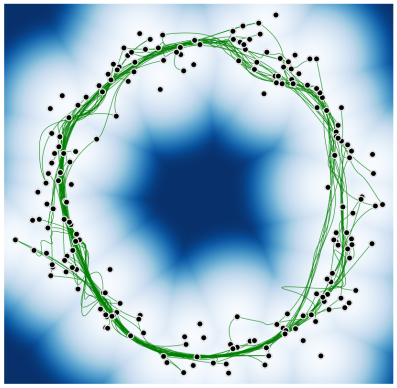
Using a VAE that decodes to products of independent vMFs (one for each bone), we formed a latent space of a walking animation in the CMU mocap dataset.[2] Fig. 8.3 visualizes this latent space with two interpolations highlighted: one by minimizing the energy proposed in Eq. (8.7), the other one linear. This figure also shows the result of decoding these two trajectories using the calibrated decoder; notice how the energy-minimized interpolation stays within the support of the data, and thus decodes to a plausible walking trajectory, unlike linear interpolation.

This experiment illustrates how our method for defining latent space geometries works even for distributions where a closed, analytical form of the KL divergence is not available. To compute the energy of the curve

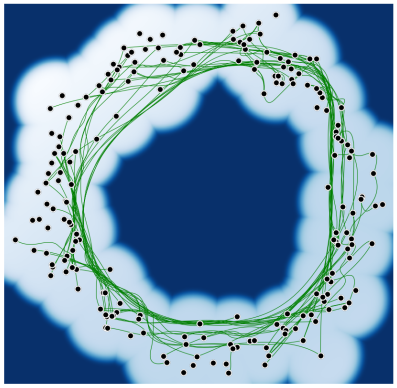
2 Sequence. `69_06` from `http://mocap.cs.cmu.edu/`, cleaned by removing some of the bones. See exact details in either the implementation or Appendix A.3.5



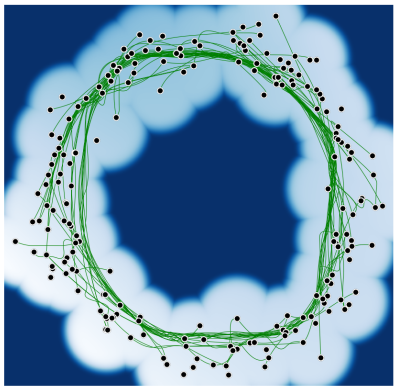
(a) Bernoulli



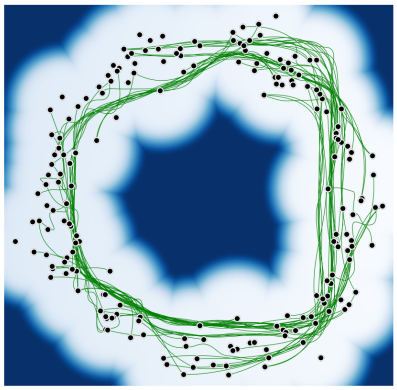
(b) Normal



(c) Beta



(d) Dirichlet



(e) Exponential

Figure 8.2: **Decoding to several distributions**. Using a toy set-up for the latent space, this figure shows the energy-minimizing curves that result from pulling back the Fisher-Rao metric when decoding to several distributions. These are colored by uncertainty, with white areas corresponding to low entropy. Most curves follow the support of the data, except for the Bernoulli decoder.

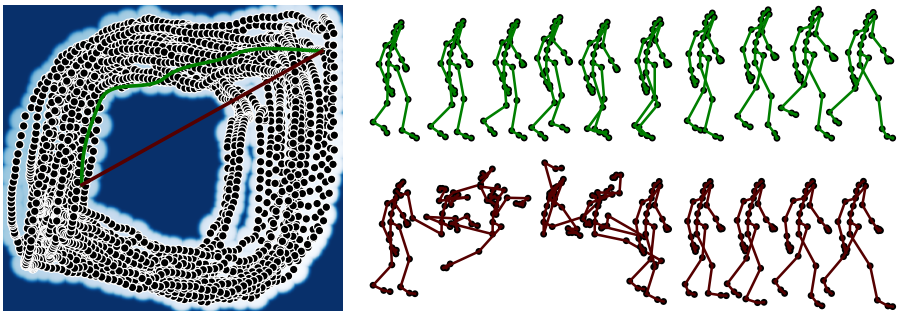


Figure 8.3: **Interpolations in a latent space of human motion**. *Left* shows the latent space of a VAE trained to decode to a product of vMF distributions, modeling the motion of a person walking by specifying the parameters of a vMF distribution for each bone in the pose. Highlighted are two interpolations: one linear in dark red, and an energy-minimizing interpolation in green. *Right* shows the result of decoding these interpolations. By staying within the support of the data, the geodesic interpolation in green produces a plausible walking animation.

presented in Fig. 8.3, we resorted to estimating the KL using samples and Monte Carlo Integration, based on the open source implementations of *hypershperical VAEs* (Davidson et al., 2018).

8.6 BLACK-BOX RANDOM GEOMETRIES, AN IMPLEMENTATION

To emphasize how general the proposed method is, this section covers how one would go around implementing this curve energy minimization in the vMF example explained in the previous section. This implementation is readily available in `stochman`, an open-source package with a toolset for manipulating stochastic manifolds and defining latent space geometries (Detlefsen et al., 2021).[3]

8.6.1 *a VAE that decodes to a product of vMFs*

The mocap data we discussed in the previous section can be distilled into a tensor of shape $b \times n_b \times 3$, where b is the number of points in the dataset (or batch size), n_b is the number of bones present in the pose, and each bone corresponds to a certain rotation in on the unit sphere in $\mathbb{R}^3$. We can implement, then, a decoder with the following form:

```
class VAE_Motion(torch.nn.Module):
    """
    A VAE that decodes to a product of vMF distributions
    """

    # We omit the initialization and other methods
    # for clarity
```

3 This section covers the vMF example script in `stochman`. In this presentation we often remove boilerplate code or the handling of batches, focusing instead on the core intuition. For a working implementation refer to the script itself.

```
    ...
    def decode(self, z) -> VonMisesFisher:
        """
        Returns the parameters mu and k of
        the decoded vMF distribution.
        """
        # A hidden layer of the decoder
        h = self.decoder(z)

        # A layer that learns the mean of the vMF
        mu = self.dec_mu(h)

        # Reshaping it to b x n_bones x 3, and normalizing
        # it so that it lives in the sphere
        batch_size, inp = mu.shape
        mu = mu.view(batch_size, inp // 3, 3)
        norm_mu = self.norm_2(mu).sqrt()
        mu = torch.div(mu, norm_mu)

        # A layer that learns the concentration of the vMF
        k = self.dec_k(h) + 0.01  # avoid collapses and singularities

        # Building a product of n_bones independent
        # distributions.
        vMF = VonMisesFisher(loc=mu, scale=k)
        return vMF
```

This VAE can be trained by minimizing the ELBO, as is usually done (see Chap. 6).

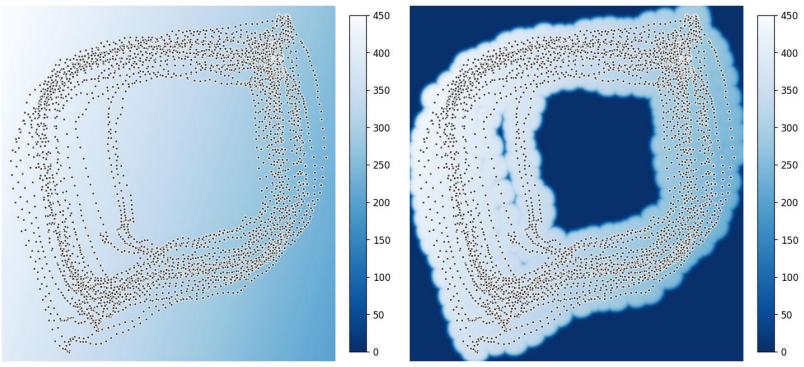
8.6.2 *Calibrating its uncertainty*

Once the VAE has been trained, the uncertainty of the decoder needs to be calibrated to define regions with low metric volume in latent space. This can be done by modifying the `decode` method we described above, and replacing the learned concentration parameter with one that corresponds to maximal uncertainty (i.e. $\kappa \to 0$).

```
class VAE_motion_with_UQ(VAE_motion):
    def decode(self, z, reweight=True) -> VonMisesFisher:
        """
        A decoder with calibrated uncertainties. It decodes to
        what it learned close to the training latent codes, and
        decodes to high uncertainty away from them.
        """
        if reweight:
            zsh = z.shape

            # Flattening extra dimensions that might appear
            # in the latent codes.
            z = z.reshape(-1, zsh[-1])

            # Getting what the network originally learned
            original_vMF_dist = super().decode(z)
            dec_mu = original_vMF_dist.loc
```



(a) Before calibration (b) After calibration

Figure 8.4: **Uncertainty calibration in the vMF example**. This figure shows the impact of calibrating the uncertainty of a VAE that decodes to a vMF. After calibration, the regions outside the support are assigned high uncertainty, where darker colors correspond to lower values of the concentration parameter.

```
            dec_k = original_vMF_dist.scale

            # Computing the distance to the support of the data
            # using the translated sigmoid.
            # 0 close to support, 1 away from it
            d_to_supp = self.translated_sigmoid(self.min_distance(z))

            # Replacing the concentration to be more
            # uncertain away from the data.
            reweighted_k = (1 - d_to_supp) * dec_k \
                + d_to_supp * (torch.ones_like(dec_k) * self.limit_k)

            # Defining a new vMF with calibrated uncertainties
            mush = dec_mu.shape
            ksh = dec_k.shape
            vMF = VonMisesFisher(
                loc=dec_mu.view(zsh[:-1] + mush[1:]),
                scale=reweighted_k.view(zsh[:-1] + ksh[1:]),
            )
        else:
            vMF = super().decode(z)

        return vMF
```

Fig. 8.4 shows the latent space illuminated by average decoded concentration before and after calibration (i.e. with `reweight` being `False` and `True` respectively in the above Python code).

8.6.3 *Statistical manifolds only need curve energy*

Once we have a calibrated decoder that outputs a vMF distribution, we can directly implement a method that computes the energy of a given curve in latent space:

```
def curve_energy(self, curve: torch.Tensor) -> torch.Tensor:
```

```
    """
    This method takes a curve of shape n_points x 2 and
    returns its (approximated) energy.
    """
    # Computing the distance between two consecutive points
    # (i.e. our delta_t)
    delta_t = (curve[1] - curve[0]).pow(2).sum()

    # Computing the distributions we will compare
    # The ones corresponding to n
    dist1 = self.decode(curve[:-1])

    # The ones corresponding to n + 1
    dist2 = self.decode(curve[1:])

    # Computing the KL divergence between them
    kl = kl_divergence(dist1, dist2)

    # Returning the approximated energy
    return 2 * (delta_t ** -1) * kl.sum()
```

The end result of this computation, the approximated energy described in Eq. (8.7), is differentiable as long as the KL divergence computed in line 18 is. This allows optimizing the curve itself. Moreover, the implementation of this curve energy is not specific to the vMF case: as long as the decoder returns a distribution with differentiable KL divergence, the curves can be optimized so as to minimize the energy. This motivates our implementation of a `StatisticalManifold` in `stochman`.[4]

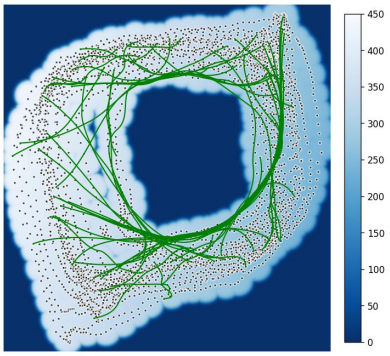


Figure 8.5: **Latent space of motion with geodesics**.

Fig. 8.5 shows 50 different geodesics obtained by minimizing the energy between randomly selected pairs of points in the latent space of our running example. The code used to compute these geodesics relies on a discretized approximation of the latent geometry, which can be easily computed using the tools present in `stochman`.[5]

8.7 DISCUSSION & LIMITATIONS

Pulling back the Fisher-Rao metric from parameter space allows us to define latent space geometries for almost any distribution. The only requirement is that the KL divergence between the distribution and itself be differentiable, which can be achieved using re-parametrization tricks and even

4 https://github.com/MachineLearningLifeScience/stochman/blob/44a18a0ae547adb84b5db6710c88980f5a9b2b23/stochman/manifold.py#L634

5 Indeed, running the example script outputs Fig. 8.5; an example of how to use this discretized approximation is available in the last lines of it.

Monte Carlo integration. This opens the doors for defining latent space geometries for tasks beyond biology (Detlefsen, Hauberg, and Boomsma, 2022), robotics (Beik-Mohammadi et al., 2021) and procedural content generation (González-Duque et al., 2022), and the computational tools are readily available in open source packages like `stochman` (Detlefsen et al., 2021).

That being said, we still face several of the limitations outlined in the previous chapter (Sec. 7.5):

1. The sensitivity to hyperparameters (like the steepness of the extrapolation to uncertainty or the number of cluster centers) is still present, and fine-tuning them is still a manual process.
2. This dependence on visual inspection forces us to work on lower dimensions, with all of our experiments being done in two dimensions.
3. The metric volume is still large *only* on the border of the support of the data. Fully safe approaches would assign "high cost" to all regions outside of the support.
4. Pulling back the Fisher-Rao for discrete distributions generates boundaries that are not tall enough. This is highlighted in our toy example (Fig. 8.2a).

The items 3 and 4 are highly relevant for decoding game content safely. In the following chapter, we will define an alternate version of this latent space geometry that addresses these two challenges by including a hierarchical layer in the decoder of the VAE.

9 GENERATING & OPTIMIZING GAME CONTENT SAFELY

This chapter describes a method for safely interpolating, sampling and optimizing content from the latent space of a tile-based Variational Autoencoder (VAE). It applies Differential Geometry and Bayesian Optimization (BO) to the problem of reliably sampling playable content from latent spaces of Deep Generative Models (DGMs) (González-Duque et al., 2022).

Chap. 6 discussed how DGMs can be used to learn an approximation of the discrete probability distribution of sequences, including tile-based video game levels like Super Mario Bros (SMB) or The Legend of Zelda. We saw, though, that these models learn only aesthetic aspects of game content, and fail to decode to playable content reliably.

Chap. 7 introduced differential geometry as a potential tool for solving this problem, and in this chapter we apply it to modify the geometry of the latent space. That way, playable content can be reliably sampled/interpolated. Additionally, a version of BO that is restricted to playable content is presented.

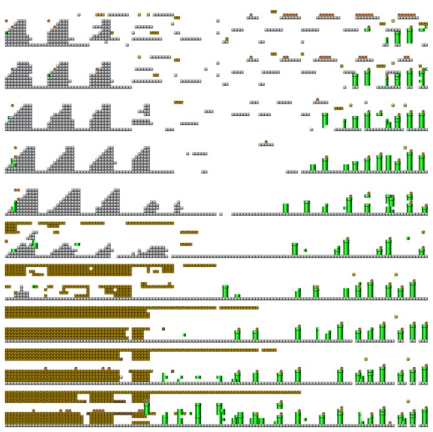
This chapter starts by motivating our approach with a hypothesis, tested empirically: playable content tends to "clump up" in only some regions of the latent space. After this, we modify the uncertainty quantification algorithms described in Sec. 7.2.3 to extrapolate to uncertainty away from playable content. This allows us to implement interpolations and diffusions/random walks that stay within playable regions. Additionally, the restricted version of BO is compared against random sampling and unrestricted BO in an optimization experiment.

This method was first proposed in (González-Duque et al., 2022) and was further expanded to optimization in a journal version that is currently under review in the *IEEE Transactions on Games* journal. This chapter covers the description, motivation, theory and results of the journal version of this paper in depth.

Throughout the chapter, we consider the terms "sampling", "random walks" and "diffusion" to be synonyms.

9.1 MOTIVATION: PLAYABLE CONTENT IN LATENT SPACE

In Chap. 6 we trained Variational Autoencoders (VAEs) on tile-based games, and on a toy example involving simple arithmetic equations. Decoding a grid of evenly-spaced latent codes and testing their levels with an artificial agent shows that the latent space has structure for its "valid" content. Fig. 9.1 visualizes this grid of SMB levels, alongside a heatmap showing whether the content was playable (blue) or not playable (white). The same



(a) Decoded grid of levels

(b) Heatmap of playability

(c) Playability heatmaps in three other VAEs

Figure 9.1: **Playability structure in latent space**. Fig. 9.1a shows a 10×10 grid of levels, which are the result of decoding evenly-spaced latent codes in the $[-5,5]^2$ square (See Fig. 6.7 for a detailed explanation). Fig. 9.1b shows a 50×50 heatmap with blue corresponding to playable levels, fading to white where the levels were not solved by Baumgarten's A* agent. There is a clear structure of playability in the latent spaces of VAEs, with a possibility of avoiding unplayable regions in interpolations, sampling and optimization. Such structure is present for different VAEs trained on the same dataset, as is shown in Fig. 9.1c.

heatmap is shown for 3 different training runs of the VAE in Fig. 9.1c. Our aim, then, is to approximate this "playability manifold", and to device algorithms for interpolation, sampling and optimization that reliably sample content from it. This would open the door to e.g. serving content directly from latent space, safely.

In the applications discussed in Chap. 9, the support of the data plays a key role in e.g. synthesizing robot movement. However, this same empirical evaluation of playability in latent space illustrates that data support and playability often do not correlate. Fig. 9.2 shows the training codes on top of the playability heatmap discussed earlier. The network extrapolates to playable levels in a neighborhood around the training codes, and there are some regions inside the support that reliably decode to unplayable content. In summary, the tools for staying within the support of the data *need to be modified*, adapted to the playability scenario.

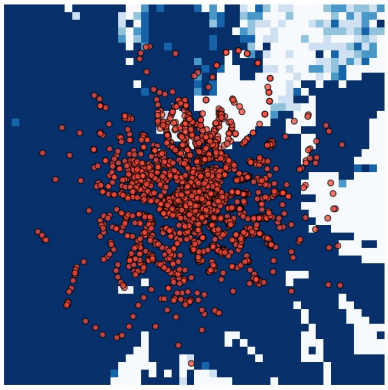


Figure 9.2: **Support and playability** do not necessarily correlate.

9.2 CALIBRATING FOR SAFETY: CHALLENGES

This section starts with a fast summary of Secs. 7.2.2 and 7.2.3.

Our methods for safe interpolation, random walks and optimization rely on modifying the latent space geometry using the tools described in Chap. 7. In particular, Sec. 7.2.3 introduced a method for modifying the geometry of the latent spaces of VAEs using Categorical decoders. This method relies on a translated sigmoid function $\alpha(\boldsymbol{z};\, \beta)$ introduced in Eq. (7.9), which we replicate here to ease the reading: if $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ are the encodings of the training set, train a K-means algorithm and arrive at K centers $\mathscr{C} = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K\}$. Define $\text{minDist}(\boldsymbol{z};\, \mathscr{C})$ be the minimum of the distances $\{\text{dist}(\boldsymbol{z}, \boldsymbol{c}_k)\}_{k=1}^{K}$, then the translated sigmoid is defined as

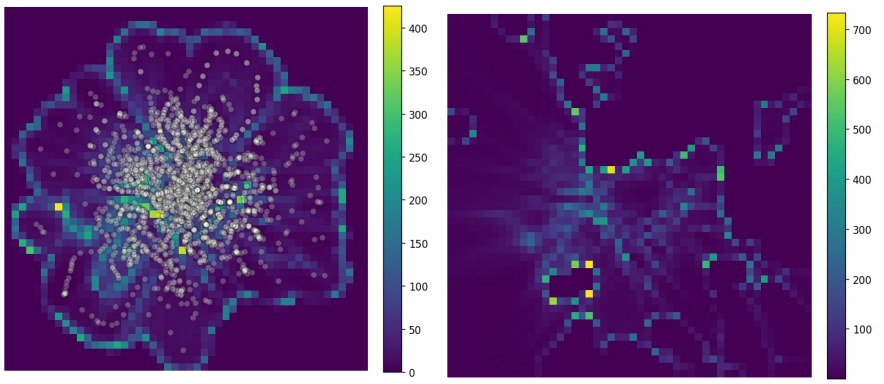
$$\alpha(\boldsymbol{z};\, \beta, \mathscr{C}) = \text{Sigmoid}\left(\frac{\text{minDist}(\boldsymbol{z};\, \mathscr{C}) - \beta s}{\beta}\right).$$

This function is approximately 0 close to the centers $\mathscr{C}$ and 1 far away. β is a hyperparameter that governs the *steepness* of this transition (see Fig. 7.4).

When using a decoder that learns a Categorical distribution over a vocabulary $\{t_1, \ldots, t_C\}$ of C tokens (i.e. $\text{dec}(\boldsymbol{z})_{l,c} \approx \text{Prob}[(\text{tile } l) = t_c]$), Detlefsen, Hauberg, and Boomsma (2022) use this translated sigmoid to extrapolate to high uncertainty away from the centers $\mathscr{C}$:

$$\widetilde{\text{dec}}(\boldsymbol{z})_{l,c} = (1 - \alpha(\boldsymbol{z};\, \mathscr{C}))\, \text{dec}(\boldsymbol{z})_{l,c} + \alpha(\boldsymbol{z};\, \mathscr{C})(1/C). \tag{9.1}$$

For the discrete case, the "wall" of high metric volume generated by this extrapolation mechanism is unsafe when interpolating, even when using our



(a) Using the original extrapolation (b) Extrapolating to $1/C$ close to unplayable.

Figure 9.3: **Metric volumes after calibration in a vanilla VAE**. Using the original extrapolation mechanism proposed by Detlefsen, Hauberg, and Boomsma results in "building a wall" of metric volume around training codes, which is shown in Fig. 9.3a. However, as can be seen in Fig. 9.2, playability and support do not necessarily correlate. We study an alternative for extrapolation, which considers unplayable codes to be highly uncertain (just like the original extrapolation treated the complement of the support). This alternative allows for assigning high cost at the boundary between the playable and unplayable parts of the latent space, but assigns 0 metric volume to all the unplayable content. By using a hierarchical layer in the decoder, we are able to assign a high cost to all unplayable codes (see Fig. 9.6b).

alternative formulation based on information geometry (see Sec. 8.4 and Fig. 8.2a). Fig. 9.3a shows the metric volume using this calibration of the decoder. Ideally, there would be high metric volume in *all* the unplayable regions of latent space instead of a wall around the support.

In summary, two challenges need to be addressed:

1. The extrapolation mechanism should affect unplayable levels instead of the complement of the support.
2. The metric volume should be high in all non-playable regions, not only at the boundary between playable and unplayable.

9.3 CALIBRATING FOR SAFETY: PLAYABLE LEVELS

To tackle the first challenge described in the previous section, the translated sigmoid function is modified: instead of having $\alpha(\boldsymbol{z}; \beta, \mathscr{C})$ be approximately 0 around the training encoding, we let it be approximately 0 close to unplayable levels, replacing $\alpha(\boldsymbol{z})$ with $(1 - \alpha(\boldsymbol{z}))$. In other words, the unplayable levels become "obstacles", places where the latent space has high volume.

This modification starts by identifying where the playable and unplayable levels are in latent space by decoding a coarse grid of size 50×50 levels and

testing these using Robin Baumgarten's A* agent.[1] These grids are visualized as heatmaps in Figs. 9.1b and 9.1c.

After decoding and simulating this grid, we have a collection of unplayable latent codes $\mathscr{U} = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_M\} \subseteq \mathcal{Z}$ scattered around the latent space. Instead of fitting K-means to the training codes, these unplayable codes are used as centers. The translated sigmoid can then be modified to extrapolate to uncertainty *close* to unplayable levels, instead of *away* from the training codes.

Mathematically, let $\text{minDist}(\boldsymbol{z};\ \mathscr{U})$ be the minimum of the distances between $\boldsymbol{z}$ and the unplayable codes in $\mathscr{U}$. With this minimum distance, the translated sigmoid $\alpha(\mu z;\ \beta, \mathscr{U})$ is approximately 0 close to unplayable levels, and approximately 1 away from unplayable levels. We modify Eq. (9.1) as

$$\widetilde{\text{dec}}(\boldsymbol{z})_{l,c} = \alpha(\boldsymbol{z};\ \mathscr{U})\,\text{dec}(\boldsymbol{z})_{i,c} + (1 - \alpha(\boldsymbol{z};\ \mathscr{U}))(1/C). \tag{9.2}$$

Fig. 9.3b shows this new extrapolation mechanism.

Still, this extrapolation procedure only assigns high metric volume to *the border* between playable and unplayable levels in latent space. Our goal is to assign a high metric volume to *all* unplayable content; that way, the unplayable levels can be cut off above a certain metric volume.

9.4 CALIBRATING FOR SAFETY: HIGH METRIC VOLUME

Fig. 9.6b shows our goal: high metric volume around all unplayable codes. To describe how this can be achieved, let's dive deeper into how our current extrapolation mechanism assigns high metric volume around training codes (or playable levels).

The output of the modified decoder in Eq. (9.2) starts being $\text{dec}(\boldsymbol{z})$ around playable codes, and gradually gets converted into a constant $1/C$ close to unplayable content. At the interface between playable and non-playable, the Jacobian of the decoder (Eq. (7.2)) achieves high values in the numerator. This translates into high values for the metric volume (Eq. (7.5)).

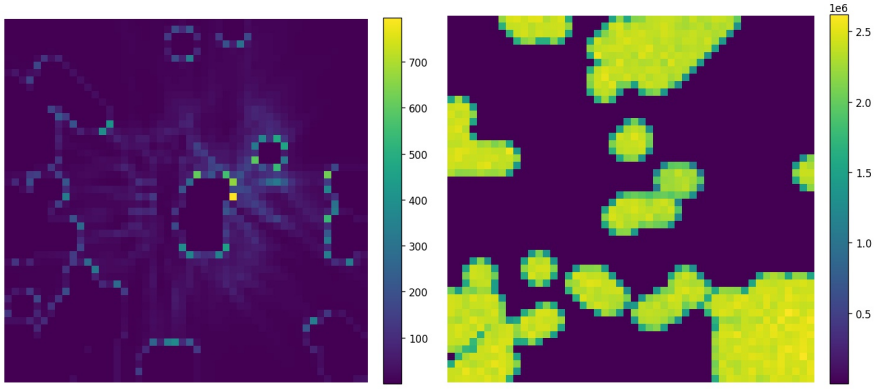
However, after this transition *there is no local change* since the decoder always returns $1/C$. In this region, the volume landscape becomes completely flat, as can be seen in Fig. 9.4c, where the unplayable regions have almost 0 volume.

Our method addresses this lack of local change around unplayable content by using a hierarchical layer in the decoder (see Fig. 9.5). Instead of decoding to a single logit for each level, our network decodes to a Normal distribution for logits. Putting this change into a mathematical form, this one-layer-hierarchical decoder is given by:

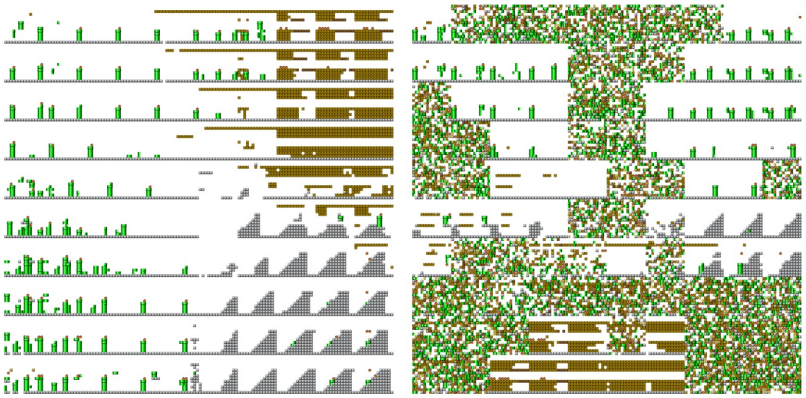
$$\text{dec}_\theta(\boldsymbol{z}) = \text{Softmax}(\boldsymbol{\mu}_\theta(\boldsymbol{z}) + \varepsilon \odot \boldsymbol{\sigma}_\theta(\boldsymbol{z})^2), \tag{9.3}$$

1 The size of this initial grid can be considered a hyperparameter. We settle for 50 and do not explore the impact of considering smaller initial grids.

(a) Playability for vanilla VAE (b) Playability for hierarchical VAE



(c) Metric volume for vanilla VAE (d) Metric volume for hierarchical VAE



(e) Grid of levels for vanilla VAE (f) Grid of levels for hierarchical VAE

Figure 9.4: **Comparing the calibration of a vanilla and a hierarchical VAE.** This figure compares two VAEs trained on SMB, without and with a hierarchical layer. Figs. 9.4a and 9.4b show a 50×50 grid of playability, where blue regions correspond to playable and white to non-playable. Using the modified decoders for the vanilla and the hierarchical alternatives, we arrive at different values for the metric volume in Figs. 9.4c and 9.4d. Notice how, while the vanilla alternative only builds a wall around unplayable content, the hierarchical alternative makes all unplayable codes expensive. This modification is also present after decoding a 10×10 grid of levels, but only for the hierarchical alternative (Figs. 9.4e and 9.4f).

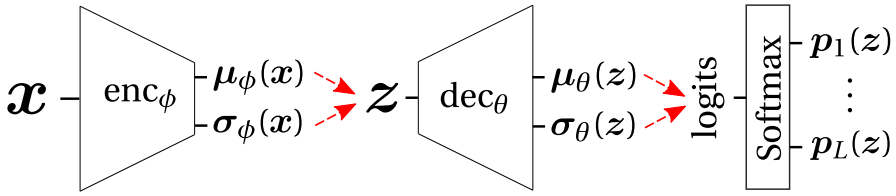


Figure 9.5: **One-layer-hierarchical VAE**. Red dashed arrows represent sampling from a Normal distribution. To be compared with Fig. 6.3.

where $\boldsymbol{\mu}_\theta$, $\boldsymbol{\sigma}_\theta$ are learned using feed-forward neural networks, and $\varepsilon \sim N(\mathbf{0}, \boldsymbol{I}_D)$ (where $D = 14 \times 14 \times 11$, the shape of the levels in logit space).

Inside this hierarchical layer, the original extrapolation mechanism proposed by Skafte, Jørgensen, and Hauberg (2019) can be leveraged (see Sec. 7.2.1). Recall that this extrapolation mechanism is devised for networks that decode to Gaussian distributions. In particular, and using the notation introduced above, the decoder is modified to be:

$$\widetilde{\text{dec}}_\theta(\boldsymbol{z}) = \text{Softmax}\left(\boldsymbol{\mu}_\theta(\boldsymbol{z}) + \epsilon \odot \widetilde{\boldsymbol{\sigma}}_\theta(\boldsymbol{z})^2\right), \tag{9.4}$$

where

$$\widetilde{\boldsymbol{\sigma}}_\theta(\boldsymbol{z}; \beta) = (1 - \alpha(\boldsymbol{z};\, \beta, \mathscr{U}))\boldsymbol{\sigma}_\theta(\boldsymbol{z}) + 10\alpha(\boldsymbol{z};\, \beta, \mathscr{U}). \tag{9.5}$$

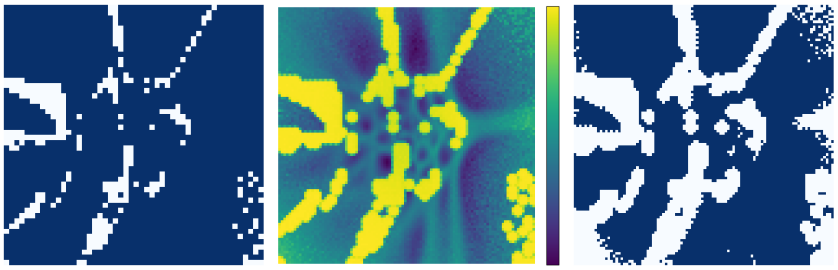
In summary, our logits will have a high variance when sampled in regions close to unplayable levels (where $\alpha(\boldsymbol{z};\, \beta, \mathscr{U}) \approx 1$), and will decode to what the network learned in the playable regions.

This high variance in the logits results in high local change, even when taking small steps in unplayable space. Figs. 9.4e and 9.4f visualize a 10×10 grid of levels, corresponding to evenly spaced latent codes, in both the previous vanilla VAE and our one-layer-hierarchical VAE. We see that for the original VAE, the extrapolated levels do not change drastically in the unplayable regions, but for the hierarchical one, the latent codes in unplayable parts decode to completely noisy logits, increasing the local distances.

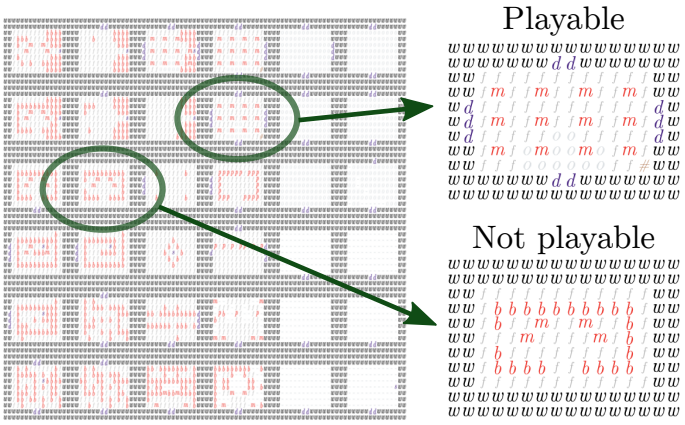
9.5 APPROXIMATING THE PLAYABILITY MANIFOLD WITH A GRAPH

After calibrating the decoder using the procedure described in the previous section, the metric volume $m(\boldsymbol{z})$ (see Eq. (7.5)) becomes a sort of "cost function" which measures playability for all $\boldsymbol{z} \in \mathscr{Z}$. Ideally, such a metric can be minimized continuously (and this is precisely the reason why we use differential geometry since it naturally allows for computing shortest paths (Lee, 2018, Chap 6.)); however, we settle for a discrete approximation of the playability regions because it provides more stable results in practice (Arvanitidis et al., 2022; Beik-Mohammadi et al., 2021). This section describes how we construct this discrete approximation.

Fig. 9.6 shows this process for the latent space of one of our Zelda models. Since we only want to consider the playable parts presented in Fig. 9.6a,



(a) Coarse grid (b) Metric volume (c) Discrete approximation



(d) Levels in latent space

Figure 9.6: **Calibrating the decoder to high volume in non-playable regions** We illustrate the process of constructing our discrete graph approximation $\mathscr{P}$ for one of our Zelda VAEs. Starting with the coarse grid approximation of how playability is distributed in latent space presented in Fig. 9.6a, we modify the decoder as described in Sec. 9.4. After this calibration, the metric volume explodes in regions close to non-playable content, as Fig. 9.6b shows. To choose only the playable levels, we consider only those that decode to metric volumes lower than a certain threshold, arriving at the discrete approximation shown in Fig. 9.6c. Finally, Fig. 9.6d shows levels in the latent space of this VAE, highlighting examples of playable and non-playable levels which align with the coarse playability grid (the example provided is not playable since it has no doors nor stairs).

(a) Coarse grid (b) $s = 0.7$ (c) $s = 0.9$ (d) $s = 1$ (e) $s = 1.1$

Figure 9.7: **Different degrees of safety when approximating**. For one of our SMB models, we show the initial grid of playability in Fig. 9.7a, a 50×50 matrix with blue blocks corresponding to playable parts of the latent space. Our framework starts with this coarse grid and builds finer discrete approximations, governed by a safety hyperparameter $s > 0$. Small values of s correspond to being safer, selecting fewer levels close to the non-playable ones. Figs. 9.7b, 9.7c, 9.7d and 9.7e show the discrete approximation in a 100×100 grid when using $s \in \{0.7, 0.9, 1, 1.1\}$ respectively. Increasing the value of s corresponds to including more levels, as can be visualized in e.g. the upper-left corner of these figures.

we consider an arbitrarily fine grid in latent space and compute the metric volume $m(\boldsymbol{z})$ for all points in this approximation. This gives us, after modifying the decoder, a heatmap like the one presented in Fig. 9.6b with non-playable levels being "far away". Our method consists of considering only the latent codes with metric volume below a certain value, leaving us with the subset of the grid corresponding to levels that are potentially playable in Fig. 9.6c. Fig. 9.6d illustrates a coarse grid of levels in latent space, highlighting examples of playable and non-playable content.

This approach comes with two perks: the grid approximation can be as fine as desired (e.g. from a 50×50 grid to a 100×100 grid), going beyond the original grid that was used to construct the first approximation. Secondly, the height of the threshold can be thought of as a "safety" hyperparameter, with cutting at lower heights representing a safer approximation. For our experimental setup we settle on a threshold of $s \cdot \mathbb{E}[2\log(m(\boldsymbol{z}))]$, where $s > 0$ is a safety hyperparameter with 0 representing choosing no levels at all, and 1 choosing those that are below the average log-squared-volume.[2] Fig. 9.6c shows the discrete graph approximation resulting from choosing $s = 1$, and Fig. 9.7 shows how this hyperparameter impacts the levels that are selected from the latent space for an SMB model.

$2\log(m(\boldsymbol{z}))$ *is just* $\log(\det(\boldsymbol{M}(\boldsymbol{z})))$, *where* $\boldsymbol{M}(\boldsymbol{z})$ *is the pullback metric (Eq.* (7.3)*).* $\det(\boldsymbol{M}(\boldsymbol{z}))$ *is known as the magnification factor.*

In summary, our modified decoder is able to assign high metric volume $m(\boldsymbol{z})$ to non-playable levels. To approximate where the playable levels are, we decode a grid as fine as desired, compute the volume $m(\boldsymbol{z})$ for all latent codes in said grid, and discard the ones above a certain value (specified by a safety hyperparameter s). This discrete grid is used to build a graph of playable content by connecting neighboring codes, and this graph is later used to compute safe interpolations and samples. This *graph of playable levels* is denoted by $\mathscr{P} \subseteq \mathscr{Z}$.

2 Log-volume is chosen for visualization since the volume usually grows to large numbers.

9.6 EXPERIMENT: INTERPOLATIONS AND RANDOM WALKS

DEFINITIONS OF FUNCTIONALITY Safety can have different meanings for different games. In the case of SMB, safety means playability, tested using Robin Baumgarten's A* agent (Sec. 6.8); in the case of Zelda, levels are defined to be playable if they pass a simple grammar check: the level (i) has either stairs or doors, (ii) has path-connected doors (iii) has doors that are complete and in the right locations, and (iv) is surrounded by walls. Two examples (one playable and one non-playable) are highlighted in Fig. 7.1 for SMB, and in Fig. 9.6d for Zelda.

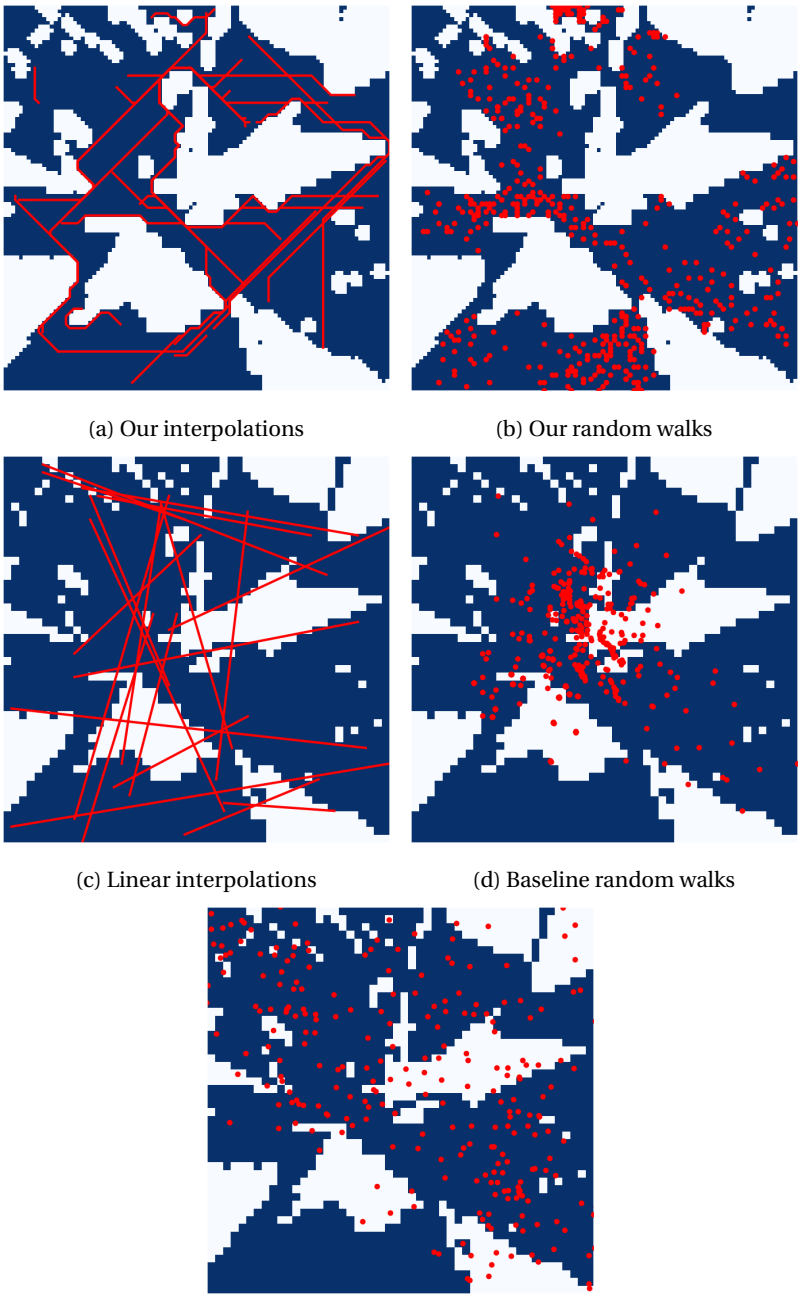
Moreover, safety does not have to relate to playability directly. Our proposed framework depends entirely on a binary classification of content and works for staying inside *one class*. An alternative definition of "safe" is explored: levels in SMB where the agent jumps at least once.

Summarizing, three definitions of functionality are tested: playability in SMB according to Baumgarten's A* agent, coherent levels in Zelda according to a simple grammar check, and levels in SMB in which Mario jumps at least once.

INTERPOLATION To interpolate between two latent codes $\boldsymbol{z}$ and $\boldsymbol{z}'$ in $\mathscr{P}$, the A* algorithm (Hart, Nilsson, and Raphael, 1968) is used. This algorithm takes a graph embedded in Euclidean space (such as $\mathscr{P}$, the graph of playable content) and computes the shortest path using a modification of Dijkstra's algorithm (Erickson, 2019, Sec. 8.6). This shortest path is found by running a search on the graph starting on $\boldsymbol{z}$, and progressively adding the neighbors and their *heuristic cost* to a priority queue. This heuristic cost includes the edge weight and the Euclidean distance between the current node and the target $\boldsymbol{z}'$. The algorithm keeps taking the minimizing element from the priority queue and iterates by adding its neighbors until arriving at the target. Examples of our A* interpolation can be found in Fig. 9.8a.

RANDOM WALKS Starting at a node $\boldsymbol{z}_0 \in \mathscr{P}$, our random walk samples uniformly from the neighbors arriving at an intermediate point $\boldsymbol{z}_0^1$. We then sample from the neighbors of this point uniformly, arriving at $\boldsymbol{z}_0^2$. This process continues for 25 intermediate, "inner" steps, until arriving at $\boldsymbol{z}_1 = \boldsymbol{z}_0^{25}$. This process continues for a given amount of "outer" steps. Different values for the number of inner steps are not explored, but it could be considered as a hyperparameter to vary in future implementations. Examples of these random walks are shown in Fig. 9.8b.

BASELINES Our graph-based interpolation and diffusion are compared against linear interpolation (which has been used in game AI settings (Schrum, Volz, and Risi, 2020)), Gaussian diffusion (i.e. sampling from a small Gaussian at each step), and a center-of-mass baseline that randomly samples a playable latent code, and takes a step in that direction. This first random



(a) Our interpolations

(b) Our random walks

(c) Linear interpolations

(d) Baseline random walks

(e) Normal random walks

Figure 9.8: **Interpolations and diffusions in the jumping regions.** This figure presents examples of interpolations and random walks for the regions of latent space that correspond to levels in which Mario jumps at least once. More precisely, Figs. 9.8a and 9.8b show the interpolations and diffusions definied in the playability graph $\mathscr{P}$. Fig. 9.8c shows example linear interpolations (used in both baselines) and Figs. 9.8d and 9.8e show the random walks of the baselines. Interpolations inside the playability graph stay away from non-functional levels (shown in white), sometimes at the cost of getting stuck bottlenecks (see the upper part of 9.8b). On the other hand, the baselines touch the regions of the latent space that correspond to non-functional content often.

walk baseline is akin to what an evolutionary algorithm would do in its exploration phases (Ha, 2017), and the second baseline can be thought of as safer since it always points towards functional content. Fig. 9.8 shows a comparison of interpolations and random walks for all the approaches when the definition of functionality is for Mario to jump at least once.

MEASURES TO COMPARE In this experiment, the playability of the decoded content is assessed. More precisely, we are interested in two quantitative measures: how "safe" is the decoded content, and how diverse is the content.

Safety was defined earlier in this section. Diversity is defined as the opposite of similarity (Boriah, Chandola, and Kumar, 2008); two levels are similar if they match in their tiles often. More precisely, given two levels $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$, their similarity is given by

$$\text{similarity}(\boldsymbol{l}_1, \boldsymbol{l}_2) = \frac{1}{wh} \sum_{i,j}^{w,h} \left[\boldsymbol{l}_1[i,j] = \boldsymbol{l}_2[i,j]\right] \tag{9.6}$$

where both levels are of size (h, w). If both levels are identical, $\text{similarity}(\boldsymbol{l}_1, \boldsymbol{l}_2) = 1$; if none of their tiles agree, their similarity is 0.

We extend this definition of similarity to collections of levels: if $\mathscr{L} = \{\boldsymbol{l}_1, \ldots, \boldsymbol{l}_M\}$ is a collection of levels with the same height and width, their similarity is given by the average of their pair-wise similarities as defined by Eq. (9.6):

$$\text{similarity}(\{\boldsymbol{l}_1, \ldots, \boldsymbol{l}_M\}) = \frac{2}{M(M-1)} \sum_{m<m'} \text{similarity}(\boldsymbol{l}_m, \boldsymbol{l}_{m'}). \tag{9.7}$$

Finally, diversity then becomes

$$\text{diversity}(\mathscr{L}) = 1 - \text{similarity}(\mathscr{L}). \tag{9.8}$$

As a reference point, the diversity of the entire collection of levels used for training in SMB and Zelda were 0.17 and 0.23 respectively.

EXPERIMENTAL SETUP Summarizing the previous sections, the methods for interpolation and random walks in the playability graph are compared against linear interpolation, sampling from Gaussians, and a center-of-mass seeking baseline. These methods are compared in terms of the playability and diversity of their decoded content. Playability has three alternate definitions, (i) Whether a Mario level was solved by an A* agent, (ii) whether a Zelda level was coherent, and (iii) whether Mario jumped at least once.

After training 10 hierarchical VAEs with different random seeds in both the SMB and Zelda databases, a grid of 50×50 evenly-spaced latent codes is decoded to levels that are measured for functionality/playability. Of the 10 VAEs trained on Zelda, 6 learned constant, non-convex or noisy representations and were thus discarded from the analysis.[3]

3 See Appendix A.3.3 for all the training details, including the visualizations of these problematic latent spaces.

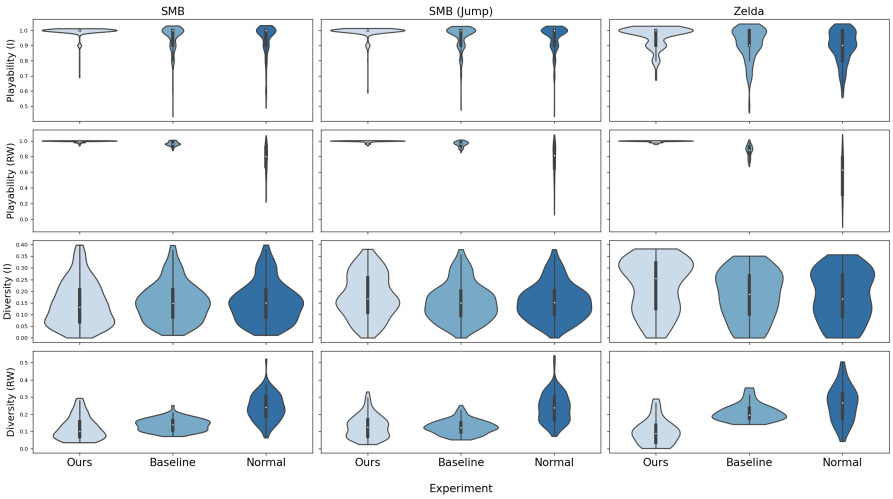


Figure 9.9: **Comparing playability and diversity.** This figure shows the distributions of playability and diversity, which are summarized in Table 9.1, where (I) stands for interpolations and (RW) stands for random walks. For each VAE we performed 20 interpolations and 10 random walks, selecting the starting points at random. These quantities were measured in each interpolation/random walk. Our interpolations and diffusions have most of their playability mass closer to 1.0 than the baselines; however, this comes at a slight cost on diversity: the mass for estimated diversities is lower than the baselines.

With this initial coarse grid of 50×50, the decoder of these hierarchical VAEs is calibrated. Then, a 100×100 grid in latent space is decoded, and the levels above the average log-squared volume $\mathbb{E}[2\log(m(\boldsymbol{z}))]$ (i.e. with a safety value of $s = 1$) are discarded. The end result of this process is a playability graph for each latent space.

Each method is tested on 20 interpolations using randomly selected start and target playable points, and 10 random walks with randomly selected starting points. For each interpolation, 10 equally spaced points were selected, and random walks take 50 steps. Playability and diversity are averaged[4] for each of these interpolations and random walks.

RESULTS Fig. 9.9 shows violin plots of the average values of playability and diversity in these interpolations and random walks. In both operations, our proposed method is highly reliable, decoding to playable content almost always (99% functionality in SMB, for example). This stands in stark contrast against e.g. the Normal baseline (taking steps with Gaussian noise). The center-seeking baseline for diffusion performs better than the Normal, but lower on average than our proposed method (0.95 vs. 0.99 respectively). These average values are shown in Table 9.1, including one standard deviation. These margins persist even when considering a dif-

4 As mentioned in Sec. 6.8, Baumgarten's agent behaved stochastically. We measured playability by averaging the individual playability of levels (1 for playable and 0 otherwise) after 5 runs.

	𝔼[playability] ↑		𝔼[diversity] ↑	
Geometry	Interpolation	Random Walks	Interpolation	Random Walks
		Super Mario Bros		
Ours	**0.993**±0.033	**0.996**±0.010	0.146±0.034	0.121±0.024
Baseline	0.953±0.084	0.963±0.026	0.154±0.028	0.138±0.026
Normal	0.949±0.093	0.773±0.169	**0.155**±0.029	**0.240**±0.026
		The Legend of Zelda		
Ours	**0.961**±0.068	**0.995**±0.011	**0.222**±0.112	0.099±0.072
Baseline	0.916±0.104	0.874±0.073	0.182±0.104	0.213±0.051
Normal	0.896±0.105	0.567±0.257	0.178±0.107	**0.261**±0.103
		Super Mario Bros (Jump)		
	𝔼[playability] ↑		𝔼[jumps > 0] ↑	
Ours	**0.990**±0.040	**0.995**±0.013	**0.99**±0.01	**1.00**±0.00
Baseline	0.957±0.078	0.960±0.034	0.90±0.03	0.75±0.08
Normal	0.952±0.083	0.768±0.200	0.90±0.02	0.94±0.02

Table 9.1: **Comparison between the proposed methods and baselines for SMB, Zelda, and the jump submanifold**. Our method is compared against the baselines on two fronts: the playability of the content decoded, as well as its diversity across the entire interpolation/random walk. Both baselines use linear interpolation, but "Baseline" corresponds to the center-seeking random walks, while "Normal" corresponds to taking Gaussian steps in latent space. This table presents the means and standard deviations after running the experiments on 10 different VAE runs for SMB, and 4 selected VAE runs for Zelda. We highlight the highest numbers per column. This shows that our proposed interpolation and random walks tend to decode to playable content more often than the baselines (indeed, the expected playability is higher for ours). These results also show that there is a trade-off between this increase in playability and the diversity of the sampled levels, especially when it comes to performing random walks on Zelda. The final third of the table also shows that the reliability holds, even when considering a different definition of functionality in SMB levels (i.e. levels in which Mario jumps at least once).

ferent notion of functionality, given by the levels in which Mario jumps at least once.

This improvement in reliability comes at a cost. The diversity of the generated content drops for our model when comparing it to the baselines. The trade-off between playability and diversity is, in our opinion, to be expected: baselines that take Normal random steps have the potential of crossing the boundaries of playability, and land at noisy levels that are diverse. This difference in diversity is not remarkable in the case of interpolations, with all methods performing about equal and with high variance.

9.7 EXPERIMENT: RESTRICTED BAYESIAN OPTIMIZATION

For a tutorial on Bayesian Optimization, see Sec. 3.3.

BAYESIAN OPTIMIZATION WITH RESTRICTED DOMAIN Finally, we test whether our approach allows for running an optimization scheme in which *every* step of the optimization is playable. Since $\mathscr{P}$ has a rough estimate of where the playable content in latent space is, restricting the acquisition function to it should render playable content. We call this approach *restricted domain* Bayesian Optimization (RBO). Furthermore, RBO is tested using three different safety hyperparameters $s \in \{1.0, 1.1, 1.3\}$ to assess its impact.

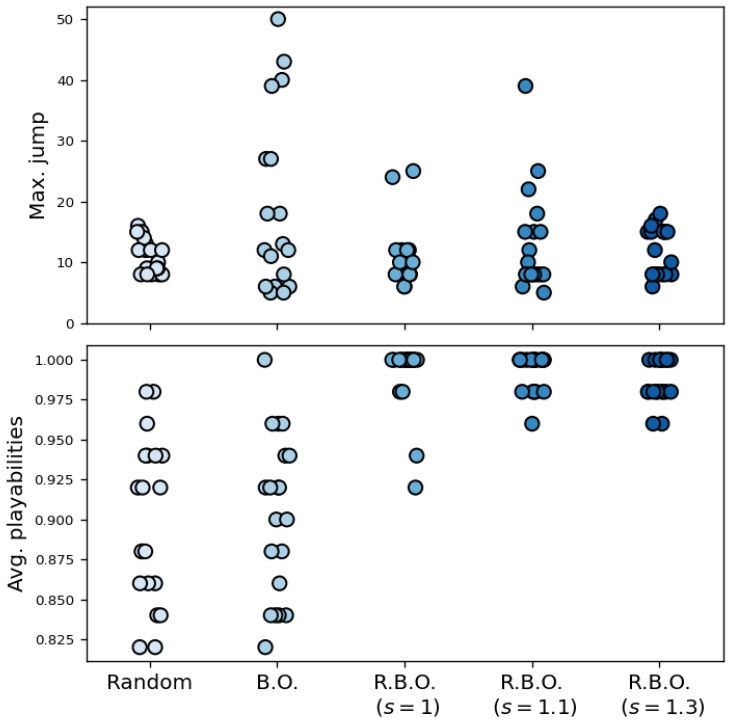
RBO is compared against vanilla BO and random sampling on two fronts: whether the levels sampled at each iteration were playable, and the quality of the optima.

EXPERIMENTAL SETUP Using one of the latent spaces trained for SMB, 20 different iterations of RBO, BO and random sampling are run, with the objective of maximizing the number of jumps[5] performed by the agent, each with a budget of 50 inner iterations/samples. Both BO methods use the default kernel provided by `botorch` (Balandat et al., 2020),[6] and the acquisition function is bounded between $[-5, 5]$ on both axes.

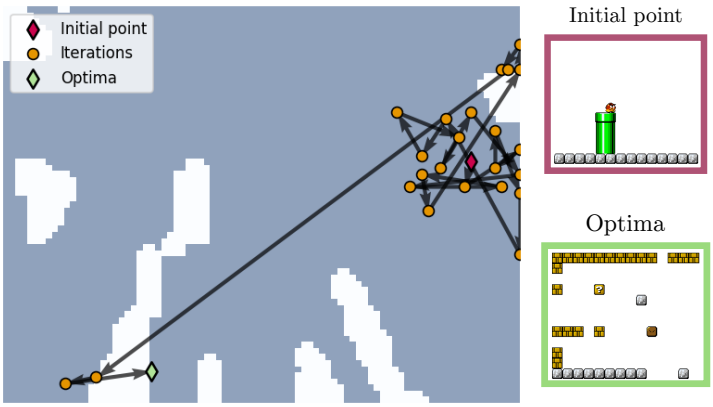
RESULTS After running 20 iterations for each of the methods we find that 16/20 runs of RBO (with $s = 1$) are *completely safe*, something that never happens for the random sampling baseline and only once for vanilla BO. This unfortunately comes at a trade-off of the quality of the optima. Fig. 9.10a shows the average playability of these 50 inner iterations for all 20 runs, as well as the maxima obtained. Vanilla BO performs best in terms of the quality of the optima, and for RBO to be competitive with random sampling, unsafer latent spaces need to be considered.

5 It is worth emphasizing that the telemetric provided by the agent and simulator are *jump action calls*, which are not exactly the same as *jump actions performed*; we aggregate the jump action calls throughout the level, which can cause extreme outliers when e.g. the agent is calling the jump action while still in the air. To be more precise, we are optimizing the agent's "intention" to jump, instead of actual jumps.

6 The exact training and model details can be found in Appendix A.2.4.



(a) **Playability and optimal jumps in restricted BO and baselines**.



(b) **An optimization trace for RBO**.

Figure 9.10: **Experiments on restricted domain Bayesian Optimization**. Fig. 9.10a illustrates how our proposed restricted domain Bayesian Optimization (RBO) compares against vanilla BO and random sampling when maximizing the number of jumps in a given level of SMB. While vanilla BO achieves better optima than the rest, RBO goes through the optimization in a safer manner (depending on the safety hyperparameter s, see Sec. 9.4). In other words, our proposed method fails to find levels with a high number of jumps in most of the traces when compared against the baseline, but it is more likely to sample playable levels in all the iterations of the optimization. We clip the y-axis of the maximum number of jumps to 50 for easier comparisons, but there were outliers for Random, BO and RBO going over 50. Fig. 9.10b shows an individual trace for RBO ($s = 1.3$). After searching the upper-right corner, the model explores the lower-left and finds an optimum with 18 jump action calls. We highlight the initial guess and the optima.

The trade-off between playability and quality of the optima is made even more evident when we start to consider higher values for s (which translate to approximations of the playability manifold that are less conservative). Although these achieve optima of better quality, the number of runs that are completely safe drops (14/20 for $s = 1.1$ and 9/20 for $s = 1.3$). This, we hypothesize, is because the objective function (amount of jumps) correlates directly with non-playability: when Baumgarten's agent gets stuck, it tends to jump in the same region until timeout.

Fig. 9.10b shows one of the 20 trajectories of our restricted BO with $s = 1.3$. The optimization process starts by searching the upper-right corner of the latent space, finding good candidates around the non-playable region. After 23 iterations, the acquisition function is maximized in the lower left, pulling the optimization towards new unexplored areas. After 25/50 iterations, an optimal candidate (with 18 jump action calls) is found.

9.8 LIMITATIONS

We see the following limitations in our approach:

ASSUMING PLAYABILITY STRUCTURE For the proposed method to work, the latent space is assumed to have structure in its playability. That is, large regions of the latent space are expected to be reliably playable. There are no theoretical guarantees for this, and it may be the case that the pattern breaks down after considering even finer grids. Fortunately, we did not run into these issues for the games we tested, and this playability structure was visible in several different training runs.

COMPUTATIONAL COST OF APPROXIMATION Relying on a discrete graph approximation for computing shortest paths and interpolations forces us to only tackle latent spaces of low dimensions. Indeed, the memory complexity of building such a grid is $O(n^d)$ where n is the grid fineness and d is the dimension of the latent space. Extending to higher dimensions should be possible by using tools from differential geometry: the computation of shortest paths on arbitrary manifolds can be performed by solving a differential equation/a continuous minimization problem (Arvanitidis et al., 2019). These tools could scale gracefully to higher dimensions (Krämer and Hennig, 2021).

COMPUTATIONAL COST OF THE COARSE GRID The initial step of computing the initial coarse grid approximation may be prohibitively expensive. Anecdotally, we saw good approximations of this latent space with fewer calls to the simulator when using a form of Active Learning in latent space: learning the boundary between playable and non-playable content using a binary classifier and querying the most uncertain points iteratively (see Uncertainty Sampling, Settles, 2009, Sec. 3.1.).

RESTRICTING HINDERS DIVERSITY AND OPTIMA Our restriction of the latent space evidently hinders metrics that correlate with unplayability and thus difficulty. As we saw in the optimization experiment (see Sec. 9.7), since jumping correlates heavily with non-playable content, the optima escape the discrete approximation. Complete safety, one of our initial desiderata, can only be achieved by highly restricting the latent space.

EVALUATION QUALITY A limitation in our evaluation is the use of Robin Baumgarten's agent. Said agent is supposed to be deterministic, but we experienced in practice that the same level could give different results when running it more than once. After private correspondence with the developer (as well as other researchers using these tools), we settled for treating it as a stochastic agent and averaging over 5 runs.

LATENT SPACE DIMENSION By using the extrapolation mechanisms described in Chap. 7, we are subject to the limitations outlined in Sec. 7.5. In particular, relying on visual inspection to set most of the hyperparameters related to the approximation of the playability graph (like β in Eq. (9.4)) limits us to latent spaces of low dimension (indeed, we only test in latent dimensions 2).

OPTIMIZATION OF THE ACQUISITION FUNCTION Our proposed safe alternative to BO (i.e. restricted domain BO) requires us to optimize the acquisition function inside a restricted subset of the original domain. We solve this optimization problem using grid search in the graph itself, and this is prohibitive in higher dimensions. Applying RBO in higher dimensions with a larger restricted domain would require using constrained optimization techniques that scale gracefully.

KERNELS FOR GRAPHS An alternative to our naïve restriction of the domain would be to construct a Gaussian Process Regression on the graph itself. Recent methods have been developed for building kernels in non-Euclidean such as graphs or Riemannian manifolds (Borovitskiy et al., 2021), and these could be leveraged for BO on these restricted settings directly (Jaquier et al., 2019).

A COMPARISON AGAINST CONSTRAINED BO We compare RBO against vanilla BO. Future work could include a comparison against a playability-aware constrained BO (Hernandez-Lobato et al., 2016), which incorporates an approximation of the playability region alongside the regression of the objective function. We would still expect such methods to sample unplayable content while they build a model of where the functional content is, but they could be safer than the vanilla alternative in, say, the second half of the optimization process.

9.9 CONCLUSION

This chapter discussed a method for safe interpolation, diffusion and optimization in the latent space of Categorical VAEs based tools from differential geometry (which were introduced in Chap. 7). Our method starts by finding out where the playable content is, decoding a coarse grid and testing it for functionality. This initial approximation is then used to calibrate the decoder and modify the latent space geometry. The resulting latent codes have high metric volume in unplayable regions, allowing us to discard those above a certain threshold (which depends on a safety hyperparameter s). The output of our method is a discrete graph, in which we define interpolations using Dijkstra's, random walks by uniformly sampling neighbors, and optimization by restricting the domain of the acquisition function.

We compare our method against simpler baselines that are commonly used by our community (like linear interpolation, or sampling from a Normal), and we see a clear trade-off: while our method is able to decode to playable content, the quality of the optima and the diversity of the generated content lowers.

Part IV

CONCLUSION

10

CONTRIBUTIONS, DISCUSSION & FUTURE WORK

This chapter summarizes the contributions, results, and discussion of the methodologies presented in parts 2 and 3 of this dissertation, including an outlook on future work.

10.1 CONTRIBUTIONS

The framework we propose (Fig. 1.1) was tested in several instances: from adapting content to planning agents in the General Video Game AI's version of Zelda in Chap. 4, to adapting Sudoku puzzles and dungeon crawler levels to human players in Chap. 5. Table 10.1 summarizes the different instances in which our framework was used.

One of the core components of the framework is a content generation algorithm, which can be specified by the designer as long as it generates levels from a vector of numbers. Chap. 6 introduced Deep Generative Models (DGMs) as a way to relax this requirement: instead of providing a content generator, the designer can provide a corpus of levels and train a DGM, which learns the distribution of the data and is able to generate novel content from said distribution.

A drawback of using DGMs for learning and generating such content automatically is that game levels have functionality requirements. Through modifying the geometry of the latent space, we were able to improve the reliability and safety of Variational Autoencoders (VAEs, a type of DGM) trained on tile-based data, like levels from the video game Super Mario Bros (Chap. 9). In the process, we developed novel tools for defining latent space geometries for almost any decoded distribution by re-framing the problem using Information Geometry in Chap. 8.

To summarize, our contributions are as follows:

1. We propose a framework for content adaption based on Bayesian Optimization, which is able to search low-dimensional design spaces efficiently, finding game content that satisfies metrics specified by the designer.
2. We tested this framework in four set-ups, using planning agents and human players.
3. By using DGMs, we relax the requirement of specifying a content generator; to assure playability we introduce a new way to modify the geometry of the latent space that penalizes *all* non-playable content.
4. Using Information Geometry and a shift of perspective from data space to parameter space, we are able to solve one of the limitations

Game	Player	Prior	Content generator	Target	Ref.
Dungeon Crawler	Planning agents	Evolved using MAP-Elites	Simple PCG generator	Win rate of 60%	Sec. 4.4
Sudoku	Human	Handcrafted	Selecting from corpus	Completion time of 180 sec.	Sec. 5.6
Dungeon Crawler	Human	Handcrafted	Selecting from corpus	Completion time of 10 sec.	Sec. 5.6
Super Mario Bros	A* agent	Constant	VAE trained on level examples	Max. number of jump actions	Sec. 9.7

Table 10.1: **Instances of the framework**. This table summarizes the components of the different instances of the framework presented in this thesis.

of previous methods for defining latent space geometries: they had only been formally defined for Variational Autoencoders (VAEs) that decode to Gaussian distributions.

5. These experiments and tools are released open source, see Sec. A.1 for links to these.

However, these contributions require more context. The next section discusses results and specifies key takeaways to be considered when applying this framework.

10.2 DISCUSSION

Circling back to the research hypotheses that guide this work (Sec. 1.1), our main hypothesis was

> Bayesian Optimization is a competitive alternative for dynamically (and safely) adapting content to users due to its sample efficiency.

We tested this hypothesis on four set-ups (Table 10.1). For the first one (planning agents using a prior built with MAP-Elites), our framework is indeed an alternative for dynamically adjusting content *as long as* the performance of the agents is not extreme.

In the presence of noisy evaluations, like the ones present in experiments involving human players, our framework performs best for games that are deterministic and have a decreasing branching factor. The only evidence we have for this claim, however, are the second and third set-ups, where we deployed our framework using hand-crafted priors in Sudoku and a dungeon crawler game respectively. Our framework performed well on Sudoku puzzles, which have no opponents, are deterministic, and have a decreasing branching factor; less so can be said about dungeon crawler levels.

What makes our framework work better for Sudoku puzzles than for Dungeon Crawler levels? We hypothesize that it is the noise present in the evaluations of the latter: the opponent AI behaved stochastically, making two runs of the same level potentially resulting in different telemetrics. Future work could explore games that are in between Sudoku and Dungeon

Crawler in terms of stochasticity and branching factor, exploring the boundaries of our framework's applicability.

That being said, the fact that our proposed framework is able to find Sudoku puzzles in a few iterations opens doors for plenty of applications. For example, a designer could take our technology and implementation[1] and deploy it in a Sudoku game which allows players to choose a desired completion time. This is made feasible by our system's simple, data-driven model of the player. The same can be hypothesized for other deterministic puzzle games.

Our second hypothesis was

> Modifying the geometry of the latent space allows for interpolating, sampling, and optimizing playable content reliably in VAEs trained on tile-based video game levels.

This hypothesis arose from the use of latent variable DGMs (more specifically, VAEs on tile-based game levels). We covered how these models are unreliable when it comes to decoding functional content in Chaps. 6 and 7.

To address this issue, we introduced a new way to modify the metric volume in latent space in Chap. 9. By approximating the playable regions of the latent space using a graph, we were able to interpolate, sample and optimize safely; however, this approximation restricts the latent space heavily, making the levels we sample less diverse, and lowering the possible maxima that can be achieved in the optimization.

In our generalization of latent space geometries to (almost) any distribution (Chap. 8) we emphasize that by "almost", we mean distributions for which Kullback-Leibler (KL) divergences can be computed and differentiated. Thankfully, this condition is quite lax: for most distributions of interest, the KL divergence is readily available or can be approximated using sampling (as long as the sampling allows to back-propagate gradients).

These geometric approaches were tested on latent spaces of only dimension 2. Therein lies a strong limitation of our approach: latent spaces of interest in games and beyond usually have latent spaces of higher dimensions, and the methods we propose rely on a visual hyperparameter tuning process that tweaks the latent geometry to a desired shape.

Still, the methods we propose are able to decode playable content more reliably than the baselines in use by the community (e.g. linear interpolation). For games that can be represented using 2D latent spaces, our methodology allows designers to build tools for safe exploration, sampling and optimization of game content. Designers could also deploy playable platformer games by stitching together playable segments *directly from latent space*.

1 Our experiments are open source. See Sec. A.1 in the appendix.

10.3 ADDRESSING LIMITATIONS & FUTURE WORK

As discussed above, the new methods proposed in this thesis can lead to future research and potential applications. This section covers how the current limitations of our systems could be addressed, and other avenues for future work.

TESTING MORE INSTANCES OF THE FRAMEWORK Table 10.1 shows the four instances of the framework that were tested in this thesis. There are several other possible instances that could be tested: on other deterministic and stochastic games, with varying branching factors. Examples include deterministic games like Sokoban or slightly stochastic games like Frogger. The tools for deploying our framework are readily available.

DEPLOYING & TESTING AT SCALE We only acquired a low amount of playtraces for the second and third instances of our framework. To control for noise and to get statistical significance, our framework could be tested on larger scales. Deploying our framework at scale should not be an issue, since the Bayesian Optimization is trained on individual play traces; aggregates of more players could be considered using contemporary GP methods that scale gracefully with the number of data points (Hensman, Matthews, and Ghahramani, 2015).

TOWARDS NON-STATIC MODELS OF THE PLAYER By including the entire playtrace of a given player in our experiments, we are implicitly stating that the player does not improve over time. Naïvely, this could be addressed by maintaining a playtrace of only the latest pairs of content specification and telemetric; a more principled approach could leverage Gaussian Processes' ability to model uncertainty: more uncertainty could be placed on earlier parts of the playtrace.

THE CASE AGAINST FLOW The keen-eyed reader might have noticed that we did not engage with the theory of *flow*, which is at the basis of Dynamic Difficulty Adjustment (DDA). Indeed, we formulate the framework as *targeting a certain telemetric* instead of *keeping the player in flow*. We argue that there is room for going *beyond flow* in the way we approach modulating difficulty since, as designers point out (Anthropy and Clark, 2014, Chap. 6), the aesthetics of some games are frustration instead of engagement.

GAMES THAT CAN ALREADY BE DEVELOPED As discussed in the previous section, there are already a couple of applications that could be built with the framework we propose: a Sudoku application that learns a data-driven model of the player and uses our framework to serve puzzles with a certain predicted completion time, an infinite platformer sampling levels

safely from latent space, and co-creative tools for exploring the *playable* regions of latent spaces of such tile-based games.

ADDING MORE INDUCTIVE BIASES TO GAUSSIAN PROCESSES In our Sudoku experiment (Sec. 5.6), we can expect that the telemetric measured (completion time) is inversely proportional to the encoding used (number of pre-filled digits). Such information could be passed to the Gaussian Process (GP) as a form of *inductive bias* (Riihimäki and Vehtari, 2010). Another potential inductive bias to include is the use of *graph kernels* (Borovitskiy et al., 2021) in our restricted Bayesian Optimization scheme (Sec. 9.7). That way, the acquisition function is naturally restricted to the playable domain.

TOWARDS GEOMETRIES IN HIGHER DIMENSIONS The experiments covered in Chaps. 8 and 9 explore defining latent geometries in low dimensions only because the uncertainty quantification of the VAEs needs to be calibrated *by hand*. We speculate that, by using models with automatic uncertainty quantification (like Laplacian Autoencoders (Miani et al., 2022), or even Gaussian Process - Latent Variable Models (Lawrence, 2003)), these latent space geometries could scale to higher dimensions.

GOING BEYOND GANS & VAES As surveyed in Chap. 6, the game AI community has focused on applications of Autoregressive Models, Generative Adversarial Networks, and Variational Autoencoders. Other Deep Generative Models (DGMs) are ripe for application in this domain (e.g. flow-based or diffusion-based models). There is a place in the literature for a survey that focuses on the affordances, applications and evaluation of DGMs in games, providing example implementations for the community. Another family of levels that could be included in this analysis are autoregressive language-based models; recent work has taken steps in this direction, implementing models that generate levels based on prompts written in English (Sudhakaran et al., 2023; Todd et al., 2023).

BRIDGING RESEARCH FIELDS The algorithms that inspire our contributions come from two fields: robotics, and biology. Our framework is an adaptation of the *Intelligent Trial-and-Error* algorithm for adapting robot gaits (Cully et al., 2015), and our algorithms for safe interpolation and extrapolation are inspired by recent advancements in robotics and protein modeling (Beik-Mohammadi et al., 2021; Detlefsen, Hauberg, and Boomsma, 2022). We see a potential for cross-collaboration between these fields: our community's focus on quality-diversity and evolutionary algorithms can be useful for protein optimization, and reliable representations of proteins can be used as inspiration for that of games (Krenn et al., 2022). In the particular case of the technology we develop, *safe* optimization could be achieved using our restricted BO (Sec. 9.7) on VAEs trained on discrete representations of molecules or proteins.

10.4 CONCLUSION

This thesis proposes and analyses a framework for adapting game content to users. Our experiments show that the use of Bayesian Optimization is a competitive alternative for adapting the content of simple, deterministic games and that a geometric approach to Deep Generative Models allows for sampling, interpolating, and optimizing safely in latent space.

The work done in this dissertation opens the doors to new research and development, from deploying our framework at scale and testing whether it would work on other instances, to improvements on algorithms for optimization in other domains.

As a final remark, it is worth emphasizing that the ideas that sparked our research came from other fields than game AI. *This is the value of interdisciplinary research.* This thesis covered a range of diverse topics, from pure mathematics (in the form of differential geometry in Chap. 8), to robotics (Chap. 7), to Experience-Driven Procedural Content Generation (Chap. 2). We see value in keeping this conversation between fields going and expect future work to focus on bringing the methods we developed to other fields, and to bring fresh ideas from other fields into game AI.

Part V

APPENDIX

A

TRAINING DETAILS & IMPLEMENTATIONS

A.1 LINKS TO OPEN SOURCE IMPLEMENTATIONS

Experiment	Ref.	URL
Example: Gaussian Processes	Chap. 3	https://github.com/miguelgondu/examples_in_thesis
Example: Bayesian Optimization vs. CMA-ES	Chap. 3	https://github.com/real-itu/benchmarking_evolution_and_bo/tree/6c8d41dfee5925d859e6060cae879e33a2fa184b/experiments/simple_comparison
Experiment: Framework on planning agents	Chap. 4	https://github.com/miguelgondu/finding_game_levels_paper
Experiment: Sudoku web application	Chap. 5	https://github.com/miguelgondu/bayesian_sudoku
Experiment: Dungeon crawler web application	Chap. 5	https://github.com/miguelgondu/bayesian_dungeoncrawler
Example: VAE trained on MNIST(1)	Sec. 6.4.6	https://github.com/miguelgondu/examples_in_thesis
Example: A minimal VAE on SMB	Sec. 6.6.1	https://github.com/miguelgondu/minimal_VAE_on_Mario
Example: A minimal VAE on Zelda	Sec. 6.6.2	https://github.com/miguelgondu/minimal_VAE_on_Mario/tree/dissertation_experiments
Example: Uncertainty and volume in MNIST(1)	Sec. 7.2.2	https://github.com/miguelgondu/examples_in_thesis
Experiment: decoding to several distributions	Sec. 8.4	https://github.com/MachineLearningLifeScience/stochman/tree/black-box-random-geometry/examples/black_box_random_geometries
Experiment: modelling human motion	Sec. 8.5	https://github.com/MachineLearningLifeScience/stochman/tree/black-box-random-geometry/examples/black_box_random_geometries
Experiment: Safely interpolating, sampling and optimizing	Chap. 9	https://github.com/miguelgondu/Mario_plays_on_a_manifold

Table A.1: **Links to open source implementations of experiments.**

A.2 TRAINING GAUSSIAN PROCESSES

A.2.1 *Running example in Chap. 3*

The running example for Chap. 3 uses `botorch`'s `SingleTaskGP` model, with a zero prior $\mu_0(x) \equiv 0$ and a Matérn$_{5/2}$ kernel. More details can be found in their documentation.[1]

A.2.2 *Intelligent trial-and-error in planning agents*

In Chap. 4 we test the *Intelligent-Trial-and-Error* algorithm, which uses MAP-Elites to build a prior for a Gaussian Process-based Bayesian Optimization.

MAP-ELITES DETAILS We ran each evolution for 10 generations, the initial amount of iterations per generation was 100, followed by 50 iterations with mutations. The behavioral descriptors are described in the Chapter (leniency, space coverage and reachability, in Sec. 4.3.2), and the performance metric's win rate w was computed using 40 rollouts. A random seed (23) was chosen for replicability.[2] This evolution was implemented using `pymelites`[3], a Python library we implemented.

BAYESIAN OPTIMIZATION We developed and used a library called `PyITaE`[4] for running Bayesian Optimization on top of priors built using MAP-Elites and `pymelites`. This library uses `GPy` to train Gaussian Processes (GPy, 2012). The kernel used was a Matérn$_{5/2}$ plus noise.[5]

A.2.3 *Gaussian Processes in Sudoku and Dungeon Crawler*

SUDOKU The prior is given by:

$$\mu_0(h) = \mu_0(h) = 600 + (h-17)(600-3)/(17-80), \tag{A.1}$$

where h is the number of hints or prefilled digits. $\mu_0(h)$ is the straight line that interpolates between $(80,3)$ and $(17,600)$. The kernel is given by an RBF plus noise. The Gaussian Processes were trained using `scikit-learn`

1 https://botorch.org/api/_modules/botorch/models/gp_regression.html#SingleTaskGP

2 The exact file used is `to_run.sh` in the repository: https://github.com/miguelgondu/finding_game_levels_paper/blob/7898a6512320f5aaaf04a2113565a3972d8545ed/to_run.sh.

3 https://github.com/miguelgondu/pymelites

4 https://github.com/miguelgondu/pyITaE

5 Script used to run the experiment: https://github.com/miguelgondu/finding_game_levels_paper/blob/7898a6512320f5aaaf04a2113565a3972d8545ed/to_run_itae.sh

instead of `PyITaE`. The acquisition function used the Expected Improvement described in Eq. (5.1), where we sample the approximation $\widetilde{f}(x)$ 10.000 times from the posterior distribution and compute the average of $\max(0, \widetilde{f}(x) - f_{\text{best}})$[6].

DUNGEON CRAWLER The prior is given by:

$$\mu_0(l, r) = (15/28)\,l + (1/4)\,r, \qquad \text{(A.2)}$$

where l and r are leniency and reachability, respectively. This is a plane that interpolates $(l = 0, r = 4, t = 1)$ and $(l = 14, r = 50, t = 20)$. The kernel used was RBF + Linear plus noise. The Gaussian Processes were trained using `scikit-learn`. The acquisition function was the Upper Confidence Bound, transformed to be optimized at the target and passed through an exponential accordingly.[7]

Tiles	Description
`#`	Floor
`-`	Void (p)
`B`	Block
`D`	Door (p)
`F`	Floor (p)
`I`	Block + Element
`L`	Block
`M`	Monster
`O`	Floor + Element (p)
`P`	Element
`S`	Stairs
`U`	Floor
`V`	Block
`W`	Wall

Table A.2: **Vocabulary in Zelda**

A.2.4 *Restricted Bayesian Optimization*

At each Bayesian Optimization loop, a single task GP model was trained using `botorch`. The prior was constant (with the value of this constant a hyperparameter to be optimized), and the kernel was a Matérn$_{5/2}$.[8]

A.3 TRAINING VARIATIONAL AUTOENCODERS

This section details the Variational Autoencoders (VAEs) that were used in our experiments. As a rule of thumb, our experiments focus on latent spaces of dimension 2, the batch size was 64, and the optimizer used was Adam w. a learning rate of 10^{-3} unless otherwise specified (Kingma and Ba, 2015). The hidden activations are all tanh, and the prior $p(\boldsymbol{z})$ is given by a unit Gaussian $N(\mathbf{0}, \boldsymbol{I}_2)$.

6 https://github.com/miguelgondu/bayesian_sudoku/blob/80d0716d494d1c5562990190467d980f5ff7318f/sudoku_experiment.py

7 https://github.com/miguelgondu/bayesian_dungeoncrawler/blob/1f29b1993945575f1af8db1e7dc31964f470d4b2/zelda_experiment.py

8 https://github.com/miguelgondu/Mario_plays_on_a_manifold/blob/dc2dc597182df6b731ef1ab3e9d4bf0572099b22/experiments/bayesian_optimization/restricted_domain_bo_on_mario_latent_space.py

A.3.1 *Example: MNIST(1)*

Details about the model, learning rates, batch sizes and epochs are available online.[9] As a short summary, our encoder and decoder were multi-layer perceptrons and had a single hidden layer of 64 neurons. Since the goal of this example is to illustrate how to modify latent space geometries, we train the network for 50 epochs without validating on a test set.

A.3.2 *Example: A minimal VAE on SMB*

This example is explained in detail in the repository linked above.[10]. To summarize, the encoder was a multi-layer perceptron with 3 hidden layers given by (512, 256, 128) nodes, before splitting into a network that predicts the mean and another network that predicts the variance, both linear layers from 128 to 2. The decoder outputs the logits of the Categorical distribution and was symmetric to the encoder.

A.3.3 *Example: VAE on The Legend of Zelda*

A minimal implementation of a VAE in *The Legend of Zelda* is available in a branch of the repository mentioned in the previous section.[11]

We found the need to reduce the learning rate to 10^{-4}. The encoder and decoders were the same as in the previous section, but note that the input size is 11×16 instead of 14×14.

A.3.4 *Experiment: decoding to several distributions*

In this experiment (Sec. 8.4), we use a toy "decoder" made of a randomly initialized neural network that maps the latent space to a parameter space. Table A.3 dives into the specifics of each distribution. An example for yet another distribution (Poisson) is available online.[12]

9 https://github.com/miguelgondu/examples_in_thesis/blob/8324bb253aafa69ccb01d0bfead5626b64f60ffc/Chap_6_and_7/vae.py

10 https://github.com/miguelgondu/minimal_VAE_on_Mario

11 https://github.com/miguelgondu/minimal_VAE_on_Mario/blob/f4ca190c103637cf22329e5d9fe047ffea49e991/zelda.py

12 https://github.com/MachineLearningLifeScience/stochman/blob/44a18a0ae547adb84b5db6710c88980f5a9b2b23/examples/black_box_random_geometries/toy_example/non_gaussian_decoder.py

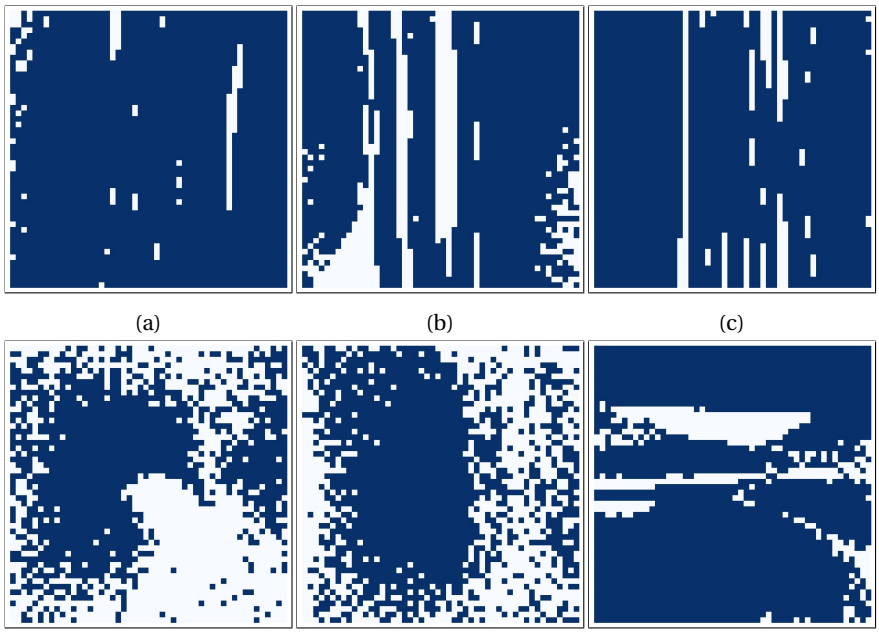


Figure A.1: **Unused latent spaces for Zelda**. In Fig. A.1a, the learned representation is constant columnwise. Figs. A.1b, A.1c and A.1f show latent spaces that are not convex, splitting the playable regions into different blocks. Finally, Figs. A.1d, and A.1e show noisy latent spaces.

Geometries for multiple distributions (Sec. 8.4)

Distribution	**Module**	**MLP**	**Seed for randomness**	β **in** $\tilde{\sigma}_\beta$
Normal	μ	Linear(2,3)	1	0.08
	σ	Linear(2,3), Softplus()		
Bernoulli	p	Linear(2,15), Sigmoid()	1	0.02
Beta	α	Linear(2,3), Softplus()	1	0.018
	β	Linear(2,3), Softplus()		
Dirichlet	α	Linear(2,3), Softplus()	17	0.018
Exponential	λ	Linear(2,3), Softplus()	17	0.018

Table A.3: **Models used in toy experiment**. This table describes the neural networks used for the experiment presented in Sec. 8.4. Following the notation of PyTorch, Linear(a,b) represents an MLP layer with a input nodes and b output nodes. In each of these networks, we calibrate the uncertainty using the methods described in Sec. 7.2, and we specify the β hyperparameter present in the translated sigmoid (Eq. (7.9)). These networks were not trained in any way: they were initialized using the provided seed.

A.3.5 *Experiment: modeling human poses*

Sec. 8.5 showcases how to define a latent space geometry when decoding to a von Mises-Fisher distribution. The implementation is available online.[13]

A.4 SAFE INTERPOLATION AND SAMPLING

The implementation of the interpolation, diffusion and optimization algorithms discussed in Chap. 9 are available online.[14]

In this experiment, we used a *one-layer-hierarchical* VAE instead. The specification of these VAEs, as well as the particular hyperparameters, are discussed in the next two paragraphs for both models.

SUPER MARIO BROS The encoder is the same as in Sec. A.3.2; however, the decoder returns the parameters of a normal distribution, i.e. there are two additional linear layers Linear($14^2, 14^2$) that learn the parameters $\mu_{\text{dec}}(\boldsymbol{z})$ and $\sigma_{\text{dec}}(\boldsymbol{z})$. These are sampled and passed through a Softmax to predict SMB levels.

THE LEGEND OF ZELDA The construction of the VAE is analogous to SMB, except for the fact that the input and output dimensions are 11×16 instead of 14×14. The learning rate used was 10^{-4}. We mention in Chap. 9 that only 4 out of the 10 VAEs trained on this dataset were used. The remaining 6 learned flat, non-convex or noisy representations, which we show in Fig. A.1.

13 https://github.com/MachineLearningLifeScience/stochman/tree/44a18a0ae547adb84b5db6710c88980f5a9b2b23/examples/black_box_random_geometries/von_mises_fisher_example

14 https://github.com/miguelgondu/Mario_plays_on_a_manifold

BIBLIOGRAPHY

Al-Roomi, Ali R. (2015). *Unconstrained Single-Objective Benchmark Functions Repository*. Halifax, Nova Scotia, Canada. URL: https://www.al-roomi.org/benchmarks/unconstrained.

Amari, Shun-ichi (1998). „Natural Gradients work Efficiently in Learning." en. In: *Neural Computation* 10.

Anthropy, Anna and Naomi Clark (2014). *A Game Design Vocabulary*. Upper Saddle River, NJ, USA: Addison-Wesley Professional. ISBN: 978-0-321-90200-5.

Arjovsky, Martin, Soumith Chintala, and Léon Bottou (2017). „Wasserstein Generative Adversarial Networks." en. In: *Proceedings of the 34th International Conference on Machine Learning*. PMLR, pp. 214–223. URL: https://proceedings.mlr.press/v70/arjovsky17a.html.

Arvanitidis, Georgios, Lars Kai Hansen, and Søren Hauberg (2018). „Latent Space Oddity: on the Curvature of Deep Generative Models." In: *International Conference on Learning Representations (ICLR)*.

Arvanitidis, Georgios, Søren Hauberg, Philipp Hennig, and Michael Schober (2019). „Fast and Robust Shortest Paths on Manifolds Learned from Data." en. In: *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, p. 10.

Arvanitidis, Georgios, Miguel González-Duque, Alison Pouplin, Dimitrios Kalatzis, and Soren Hauberg (2022). „Pulling back information geometry." en. In: *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 4872–4894. URL: https://proceedings.mlr.press/v151/arvanitidis22b.html.

Awiszus, Maren, Frederik Schubert, and Bodo Rosenhahn (2020). „TOAD-GAN: Coherent Style Level Generation from a Single Example." en. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 16.1, 10–16. ISSN: 2334-0924, 2326-909X. DOI: 10.1609/aiide.v16i1.7401.

— (2021). „World-GAN: a Generative Model for Minecraft Worlds." en. In: *2021 IEEE Conference on Games (CoG)*. Copenhagen, Denmark: IEEE, 1–8. ISBN: 978-1-66543-886-5. DOI: 10.1109/CoG52621.2021.9619133. URL: https://ieeexplore.ieee.org/document/9619133/.

Bakkes, Sander, Shimon Whiteson, Guangliang Li, George Viorel Visniuc, Efstathios Charitos, Norbert Heijne, and Arjen Swellengrebel (2014). „Challenge balancing for personalised game spaces." en. In: *2014 IEEE Games Media Entertainment*. Toronto, ON: IEEE, 1–8. ISBN: 978-1-4799-7545-7. DOI: 10.1109/GEM.2014.7047971. URL: http://ieeexplore.ieee.org/document/7047971/.

Balandat, Maximilian, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy (2020). „BoTorch:

A Framework for Efficient Monte-Carlo Bayesian Optimization.“ In: *Advances in Neural Information Processing Systems 33*. URL: http://arxiv.org/abs/1910.06403.

Ballard, Dana H. (1987). In: *Proceedings of the Sixth National Conference on Artificial Intelligence*.

Bamford, Chris, Shengyi Huang, and Simon Lucas (2022). „Griddly: A platform for AI research in games.“ In: 2011.06363. arXiv:2011.06363 [cs]. URL: http://arxiv.org/abs/2011.06363.

Bamford, Christopher, Minqi Jiang, Mikayel Samvelyan, and Tim Rocktäschel (2022). „GriddlyJS: A Web IDE for Reinforcement Learning.“ In: *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. URL: https://openreview.net/forum?id=YmacJv0i_UR.

Beik-Mohammadi, Hadi, Søren Hauberg, Georgios Arvanitidis, Gerhard Neumann, and Leonel Rozo (2021). „Learning Riemannian Manifolds for Geodesic Motion Skills.“ en. In: *Robotics: Science and Systems XVII*. Robotics: Science and Systems Foundation. ISBN: 978-0-9923747-7-8. DOI: 10.15607/RSS.2021.XVII.082. URL: http://www.roboticsproceedings.org/rss17/p082.pdf.

Bengio, Yoshua, Aaron Courville, and Pascal Vincent (2014). „Representation Learning: A Review and New Perspectives.“ In: arXiv:1206.5538. arXiv:1206.5538 [cs]. URL: http://arxiv.org/abs/1206.5538.

Bingham, Derek (2013). *Optimization Test Functions and Datasets*. Accessed: 20-03-2023. URL: https://www.sfu.ca/~ssurjano/optimization.html.

Binois, Mickaël and Nathan Wycoff (2022). „A Survey on High-dimensional Gaussian Process Modeling with Application to Bayesian Optimization.“ en. In: *ACM Transactions on Evolutionary Learning and Optimization* 2.2, 1–26. ISSN: 2688-299X, 2688-3007. DOI: 10.1145/3545611.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag. ISBN: 0387310738.

Blei, David M., Alp Kucukelbir, and Jon D. McAuliffe (2017). „Variational Inference: A Review for Statisticians.“ en. In: *Journal of the American Statistical Association* 112.518, 859–877. ISSN: 0162-1459, 1537-274X. DOI: 10.1080/01621459.2017.1285773.

Bond-Taylor, Sam, Adam Leach, Yang Long, and Chris G. Willcocks (2022). „Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models.“ In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.11. arXiv:2103.04922 [cs, stat], 7327–7347. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2021.3116668.

Boriah, Shyam, Varun Chandola, and Vipin Kumar (2008). „Similarity Measures for Categorical Data: A Comparative Evaluation.“ In: *Proceedings of the 2008 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, 243–254. ISBN: 978-0-89871-654-

2. DOI: 10.1137/1.9781611972788.22. URL: https://epubs.siam.org/doi/10.1137/1.9781611972788.22.

Borovitskiy, Viacheslav, Iskander Azangulov, Alexander Terenin, Peter Mostowsky, Marc Deisenroth, and Nicolas Durrande (2021). „Matérn Gaussian Processes on Graphs." en. In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. PMLR, 2593–2601. URL: https://proceedings.mlr.press/v130/borovitskiy21a.html.

Browne, Cameron B., Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton (2012). „A Survey of Monte Carlo Tree Search Methods." en. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1, pp. 1–43. ISSN: 1943-068X, 1943-0698. DOI: 10.1109/TCIAIG.2012.2186810.

Capps, Benjamin and Jacob Schrum (2021). „Using multiple generative adversarial networks to build better-connected levels for mega man." en. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Lille France: ACM, 66–74. ISBN: 978-1-4503-8350-9. DOI: 10.1145/3449639.3459323. URL: https://dl.acm.org/doi/10.1145/3449639.3459323.

Cobbe, Karl, Christopher Hesse, Jacob Hilton, and John Schulman (2020). „Leveraging Procedural Generation to Benchmark Reinforcement Learning." In: arXiv:1912.01588. arXiv:1912.01588 [cs, stat]. URL: http://arxiv.org/abs/1912.01588.

Cooper, Seth (2022). „Sturgeon: Tile-Based Procedural Level Generation via Learned and Designed Constraints." en. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 18.11, 26–36. ISSN: 2334-0924. DOI: 10.1609/aiide.v18i1.21944.

Cooper, Seth and Anurag Sarkar (2020). „Pathfinding Agents for Platformer Level Repair." In: *Proceedings of the Experimental AI in Games (EXAG) Workshop at AIIDE*.

Cully, Antoine, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret (2015). „Robots that can adapt like animals." In: *Nature* 521.7553, pp. 503–507. ISSN: 1476-4687. DOI: 10.1038/nature14422.

Davidson, Tim R., Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak (2018). „Hyperspherical Variational Auto-Encoders." In: *34th Conference on Uncertainty in Artificial Intelligence (UAI-18)*.

Deisenroth, Marc Peter, Yicheng Luo, and Mark van der Wilk (2020). *A Practical Guide to Gaussian Processes*. Accessed: 20-03-2023. URL: https://infallible-thompson-49de36.netlify.app/.

Demediuk, Simon, Marco Tamassia, Xiaodong Li, and William L. Raffe (2019). „Challenging AI: Evaluating the Effect of MCTS-Driven Dynamic Difficulty Adjustment on Player Enjoyment." In: *ACM International Conference Proceeding Series*. ISSN: 9781450366038. DOI: 10.1145/3290688.3290748.

Detlefsen, Nicki S., Alison Pouplin, Cilie W. Feldager, Cong Geng, Dimitris Kalatzis, Helene Hauschultz, Miguel González-Duque, Frederik Warburg, Marco Miani, and Søren Hauberg (2021). „StochMan." In: *GitHub. Note: https://github.com/MachineLearningLifeScience/stochman/*.

Detlefsen, Nicki Skafte, Søren Hauberg, and Wouter Boomsma (2022). „Learning meaningful representations of protein sequences." en. In: *Nature Communications* 13.1, p. 1914. ISSN: 2041-1723. DOI: 10.1038/s41467-022-29443-w.

Dhariwal, Prafulla and Alexander Nichol (2021). „Diffusion models beat GANs on image synthesis." In: *Advances in neural information processing systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., pp. 8780–8794. URL: https://proceedings.neurips.cc/paper/2021/file/49ad23d1ec9fa4bd8d77d02681df5cfa-Paper.pdf.

Duvenand, David (2014). *The Kernel Cookbook: Advice on Covariance functions*. Accessed: 20-03-2023. URL: https://www.cs.toronto.edu/~duvenaud/cookbook/.

Earle, Sam, Justin Snider, Matthew C. Fontaine, Stefanos Nikolaidis, and Julian Togelius (2021). „Illuminating Diverse Neural Cellular Automata for Level Generation." In: *arXiv:2109.05489 [cs]*. arXiv: 2109.05489. URL: http://arxiv.org/abs/2109.05489.

Edwards, Maria, Ming Jiang, and Julian Togelius (2021). „Search-Based Exploration and Diagnosis of TOAD-GAN." en. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 17.1, 140–147. ISSN: 2334-0924, 2326-909X. DOI: 10.1609/aiide.v17i1.18901.

Eklund, David and Søren Hauberg (2019). „Expected path length on random manifolds." In: *arXiv preprint*.

Erickson, Jeff (2019). *Algorithms*. URL: http://algorithms.wtf.

Eriksson, David, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek (2019). „Scalable Global Optimization via Local Bayesian Optimization." In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc. URL: https://papers.nips.cc/paper/2019/hash/6c990b7aca7bc7058f5e98ea909e924b-Abstract.html.

Fontaine, Matthew C., Julian Togelius, Stefanos Nikolaidis, and Amy K. Hoover (2020). „Covariance matrix adaptation for the rapid illumination of behavior space." en. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. Cancún Mexico: ACM, 94–102. ISBN: 978-1-4503-7128-5. DOI: 10.1145/3377930.3390232. URL: https://dl.acm.org/doi/10.1145/3377930.3390232.

GPy (2012). *GPy: A Gaussian process framework in python*. http://github.com/SheffieldML/GPy.

Gage, Zach and Jack Schlesinger. *Good Sudoku*. https://www.playgoodsudoku.com/. Accessed: 17-01-2023.

Gardner, Jacob R, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson (2018). „GPyTorch: Blackbox Matrix-Matrix Gaus-

sian Process Inference with GPU Acceleration.“ In: *Advances in Neural Information Processing Systems*.

Giacomello, Edoardo, Pier Luca Lanzi, and Daniele Loiacono (2018). „DOOM Level Generation Using Generative Adversarial Networks.“ In: *2018 IEEE Games, Entertainment, Media Conference (GEM)*, pp. 316–323. DOI: 10.1109/GEM.2018.8516539.

González-Duque, Miguel, Rasmus Berg Palm, and Sebastian Risi (2021). „Fast Game Content Adaptation Through Bayesian-based Player Modelling.“ en. In: *2021 IEEE Conference on Games (CoG)*. Copenhagen, Denmark: IEEE, pp. 01–08. ISBN: 978-1-66543-886-5. DOI: 10.1109/CoG52621.2021.9619018. URL: https://ieeexplore.ieee.org/document/9619018/.

González-Duque, Miguel, Rasmus Berg Palm, David Ha, and Sebastian Risi (2020). „Finding Game Levels with the Right Difficulty in a Few Trials through Intelligent Trial-and-Error.“ In: *2020 IEEE Conference on Games (CoG)*, pp. 503–510. DOI: 10.1109/CoG47356.2020.9231548.

González-Duque, Miguel, Rasmus Berg Palm, Søren Hauberg, and Sebastian Risi (2022). „Mario Plays on a Manifold: Generating Functional Content in Latent Space through Differential Geometry.“ In: *2022 IEEE Conference on Games (CoG)*, pp. 385–392. DOI: 10.1109/CoG51982.2022.9893612.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). „Generative Adversarial Nets.“ In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc. URL: https://papers.nips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html.

Gutierrez, Jake and Jacob Schrum (2020). „Generative Adversarial Network Rooms in Generative Graph Grammar Dungeons for The Legend of Zelda.“ In: *2020 IEEE Congress on Evolutionary Computation (CEC)*, 1–8. DOI: 10.1109/CEC48606.2020.9185631.

Gómez-Bombarelli, Rafael, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik (2018). „Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules.“ en. In: *ACS Central Science* 4.2, 268–276. ISSN: 2374-7943, 2374-7951. DOI: 10.1021/acscentsci.7b00572.

Görtler, Jochen, Rebecca Kehlbeck, and Oliver Deussen (2019). „A Visual Exploration of Gaussian Processes.“ en. In: *Distill* 4.4, e17. ISSN: 2476-0757. DOI: 10.23915/distill.00017.

Ha, David (2017). „A Visual Guide to Evolution Strategies.“ In: *blog.otoro.net*. URL: https://blog.otoro.net/2017/10/29/visual-evolution-strategies/.

Hansen, N. and A. Ostermeier (1996). „Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adap-

tation.“ In: *Proceedings of IEEE International Conference on Evolutionary Computation*, 312–317. DOI: 10.1109/ICEC.1996.542381.

Hao, Ya'nan, Suoju He, Junping Wang, Xiao Liu, jiajian Yang, and Wan Huang (2010). „Dynamic Difficulty Adjustment of Game AI by MCTS for the game Pac-Man.“ In: *2010 Sixth International Conference on Natural Computation*. Vol. 8, 3918–3922. DOI: 10.1109/ICNC.2010.5584761.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). „A Formal Basis for the Heuristic Determination of Minimum Cost Paths.“ In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

Hauberg, Søren (2022). „Differential geometry for generative modeling.“ en. In: URL: http://www2.compute.dtu.dk/~sohau/weekendwithbernie.

Hello Games (2016). *No Man's Sky (Press Kit)*. Accessed: 23-03-2023. URL: https://www.nomanssky.com/press/#features.

Hennig, Philipp, Michael A. Osborne, and Hans P. Kersting (2022). *Probabilistic Numerics: Computation as Machine Learning*. Cambridge University Press. DOI: 10.1017/9781316681411.

Hensman, James, Alexander Matthews, and Zoubin Ghahramani (2015). „Scalable Variational Gaussian Process Classification.“ In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Guy Lebanon and S. V. N. Vishwanathan. Vol. 38. Proceedings of Machine Learning Research. San Diego, California, USA: PMLR, pp. 351–360. URL: https://proceedings.mlr.press/v38/hensman15.html.

Hernandez-Lobato, Jose Miguel, Michael A Gelbart, Ryan P Adams, Matthew W Hoffman, and Zoubin Ghahramani (2016). „A General Framework for Constrained Bayesian Optimization using Information-based Search.“ en. In: *Journal of Machine Learning Research*, p. 53.

Higgins, Chris (2014). *No Man's Sky would take 5 billion years to explore*. Accessed: 23-03-2023. URL: https://www.wired.co.uk/article/no-mans-sky-planets.

Hoogeboom, Emiel, Jorn Peters, Rianne van den Berg, and Max Welling (2019). „Integer Discrete Flows and Lossless Compression.“ In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2019/hash/9e9a30b74c49d07d8150c8c83b1ccf07-Abstract.html.

Hoogeboom, Emiel, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling (2021). „Argmax flows and multinomial diffusion: Learning categorical distributions.“ In: *Advances in neural information processing systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., pp. 12454–12465. URL: https://proceedings.neurips.cc/paper/2021/file/67d96d458abdef21792e6d8e590244e7-Paper.pdf.

Hunicke, Robin (2005). „The case for dynamic difficulty adjustment in games.“ en. In: *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology - ACE '05*. Valencia,

Spain: ACM Press, 429–433. ISBN: 978-1-59593-110-8. DOI: 10.1145/1178477.1178573. URL: http://portal.acm.org/citation.cfm?doid=1178477.1178573.

Hunicke, Robin and Vernell Chapman (2004). „AI for dynamic difficulty adjustment in games.“ In: *AAAI Workshop - Technical Report* WS-04-04, 91–96.

Irfan, Ayesha, Adeel Zafar, and Shahbaz Hassan (2019). „Evolving Levels for General Games Using Deep Convolutional Generative Adversarial Networks.“ In: *2019 11th Computer Science and Electronic Engineering (CEEC)*, 96–101. DOI: 10.1109/CEEC47804.2019.8974332.

Jain, Rishabh, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius (2016). „Autoencoders for Level Generation, Repair, and Recognition.“ In: *Proceedings of the ICCC Workshop on Computational Creativity and Games*. Association for Computational Creativity.

Jaquier, N, L. Rozo, S. Calinon, and M. Bürger (2019). „Bayesian Optimization meets Riemannian Manifolds in Robot Learning.“ In: *In Proc of the Conference on Robot Learning (CoRL)*. Osaka, Japan.

Jaquier, Noémie and Tamim Asfour (2022). „Riemannian geometry as a unifying theory for robot motion learning and control.“ In: *International Symposium on Robotics Research (ISRR)*. Blue Sky. Geneva, Switzerland.

Jaquier N. Borovitskiy, V., A. Smolensky, A. Terenin, T. Asfour, and L. Rozo (2021). „Geometry-aware Bayesian Optimization in Robotics using Riemannian Matérn Kernels.“ In: *Conference on Robot Learning (CoRL)*.

Jennings-Teats, Martin, Gillian Smith, and Noah Wardrip-Fruin (2010). „Polymorph: dynamic difficulty adjustment through level generation.“ en. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. Monterey California: ACM, 1–4. ISBN: 978-1-4503-0023-0. DOI: 10.1145/1814256.1814267. URL: https://dl.acm.org/doi/10.1145/1814256.1814267.

Karpinskyj, Stephen, Fabio Zambetta, and Lawrence Cavedon (2014). „Video game personalisation techniques: A comprehensive survey.“ en. In: *Entertainment Computing* 5.4, 211–218. ISSN: 18759521. DOI: 10.1016/j.entcom.2014.09.002.

Karras, Tero, Samuli Laine, and Timo Aila (2019). „A Style-Based Generator Architecture for Generative Adversarial Networks.“ In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4396–4405. DOI: 10.1109/CVPR.2019.00453.

Karras, Tero, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila (2020). „Analyzing and Improving the Image Quality of StyleGAN.“ en. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE, 8107–8116. ISBN: 978-1-72817-168-5. DOI: 10.1109/CVPR42600.2020.00813. URL: https://ieeexplore.ieee.org/document/9156570/.

Karth, Isaac, Batu Aytemiz, Ross Mawhorter, and Adam M. Smith (2021). „Neurosymbolic Map Generation with VQ-VAE and WFC.“ en. In: *The 16th International Conference on the Foundations of Digital Games (FDG)*

2021. Montreal QC Canada: ACM, 1–6. ISBN: 978-1-4503-8422-3. DOI: 10.1145/3472538.3472584. URL: https://dl.acm.org/doi/10.1145/3472538.3472584.

Kaushik, Rituraj, Pierre Desreumaux, and Jean-Baptiste Mouret (2020). „Adaptive Prior Selection for Repertoire-Based Online Adaptation in Robotics." In: *Frontiers in Robotics and AI* 6, p. 151. ISSN: 2296-9144. DOI: 10.3389/frobt.2019.00151.

Khajah, Mohammad M. (2017). „Optimizing Game Engagement via Nonparametric Models and Manipulations of Difficulty, Tension, and Perceived Performance." PhD thesis. ProQuest Dissertations Publishing: University of Colorado at Boulder.

Khajah, Mohammad M., Brett D. Roads, Robert V. Lindsey, Yun-En Liu, and Michael C. Mozer (2016). „Designing Engaging Games Using Bayesian Optimization." en. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. San Jose California USA: ACM, 5571–5582. ISBN: 978-1-4503-3362-7. DOI: 10.1145/2858036.2858253. URL: https://dl.acm.org/doi/10.1145/2858036.2858253.

Kim, Seung Wook, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler (2020). „Learning to Simulate Dynamic Environments With GameGAN." en. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE, 1228–1237. ISBN: 978-1-72817-168-5. DOI: 10.1109/CVPR42600.2020.00131. URL: https://ieeexplore.ieee.org/document/9156900/.

Kingma, Diederik P. and Jimmy Ba (2015). „Adam: A Method for Stochastic Optimization." In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1412.6980.

Kingma, Diederik P. and Max Welling (2014). „Auto-Encoding Variational Bayes." en. In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=33X9fd2-9FyZd.

Kitfox Games (2003). *Dwarf Fortress (Press Kit)*. Accessed: 28-03-2023. URL: https://www.kitfoxgames.com/press/dwarf_fortress/index.html.

Krenn, Mario et al. (2022). „SELFIES and the future of molecular string representations." en. In: *Patterns* 3.10, p. 100588. ISSN: 26663899. DOI: 10.1016/j.patter.2022.100588.

Kristensen, Jeppe Theiss, Arturo Valdivia, and Paolo Burelli (2020). „Estimating Player Completion Rate in Mobile Puzzle Games Using Reinforcement Learning." In: *2020 IEEE Conference on Games (CoG)*, pp. 636–639. DOI: 10.1109/CoG47356.2020.9231581.

Krämer, Nicholas and Philipp Hennig (2021). „Linear-Time Probabilistic Solution of Boundary Value Problems." In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 11160–11171. URL: https://proceedings.neurips.cc/paper/2021/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html.

Kumaran, Vikram, Bradford Mott, and James Lester (2019). „Generating Game Levels for Multiple Distinct Games with a Common Latent Space.“ en. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 15.11, 102–108. ISSN: 2334-0924.

Lawrence, Neil (2003). „Gaussian Process Latent Variable Models for Visualisation of High Dimensional Data.“ In: *Advances in Neural Information Processing Systems*. Vol. 16. MIT Press. URL: https://proceedings.neurips.cc/paper/2003/hash/9657c1fffd38824e5ab0472e022e577e-Abstract.html.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). „Gradient-based learning applied to document recognition.“ In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791.

Lee, John M. (2000). *Introduction to Smooth Manifolds*. Springer.

— (2018). *Introduction to Riemannian Manifolds*. Springer.

Li, Xinyu, Suoju He, Yue Dong, Qing Liu, Xiao Liu, Yiwen Fu, Zhiyuan Shi, and Wan Huang (2010). „To create DDA by the approach of ANN from UCT-created data.“ In: *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*. Vol. 8, pp. V8–475–V8–478. DOI: 10.1109/ICCASM.2010.5620008.

Liu, Jialin, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N. Yannakakis, and Julian Togelius (2020). „Deep learning for procedural content generation.“ en. In: *Neural Computing and Applications* 33.1, pp. 19–37. ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-020-05383-8.

Liu, Xiao, Yao Li, Suoju He, Yiwen Fu, Jiajian Yang, Donglin Ji, and Yang Chen (2009). „To Create Intelligent Adaptive Game Opponent by Using Monte-Carlo for the Game of Pac-Man.“ In: *2009 Fifth International Conference on Natural Computation*. Vol. 5, 598–602. DOI: 10.1109/ICNC.2009.633.

Makarova, Anastasia, Huibin Shen, Valerio Perrone, Aaron Klein, Jean Baptiste Faddoul, Andreas Krause, Matthias Seeger, and Cedric Archambeau (2022). „Automatic Termination for Hyperparameter Optimization.“ en. In: *Proceedings of the First International Conference on Automated Machine Learning*. PMLR, pp. 7/1–21. URL: https://proceedings.mlr.press/v188/makarova22a.html.

Martens, James (2020). „New insights and perspectives on the natural gradient method.“ In: *Journal of Machine Learning Research* 21.146. Citation Key: JMLR:v21:17-678, pp. 1–76.

Maus, Natalie, Haydn Thomas Jones, Juston Moore, Matt Kusner, John Bradshaw, and Jacob R. Gardner (2022). „Local Latent Space Bayesian Optimization over Structured Inputs.“ In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho. URL: https://openreview.net/forum?id=nZRTRevUO-.

Miani, Marco, Frederik Warburg, Pablo Moreno-Muñoz, Nicki Skafte Detlefsen, and Søren Hauberg (2022). „Laplacian Autoencoders for Learning

Stochastic Representations.“ In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Mojang Studios (2011). *Minecraft (Press Kit)*. Accessed: 28-03-2023. URL: https://www.igdb.com/games/minecraft/presskit.

Moon, Hee-Seung and Jiwon Seo (2020). „Dynamic Difficulty Adjustment via Fast User Adaptation.“ en. In: *Adjunct Publication of the 33rd Annual ACM Symposium on User Interface Software and Technology*. Virtual Event USA: ACM, 13–15. ISBN: 978-1-4503-7515-3. DOI: 10.1145/3379350.3418578. URL: https://dl.acm.org/doi/10.1145/3379350.3418578.

Mossmouth and BlitWorks (2020). *Spelunky 2 (Press Kit)*. Accessed: 28-03-2023. URL: https://www.igdb.com/games/spelunky-2/presskit.

Mouret, Jean-Baptiste and Jeff Clune (2015). „Illuminating search spaces by mapping elites.“ In: arXiv:1504.04909. arXiv:1504.04909 [cs, q-bio] Citation Key: Mouret:MAP-Elites:2015. DOI: 10.48550/arXiv.1504.04909. URL: http://arxiv.org/abs/1504.04909.

Nielsen, Frank (2020). „An Elementary Introduction to Information Geometry.“ In: *Entropy* 22.10. ISSN: 1099-4300. DOI: 10.3390/e22101100. URL: https://www.mdpi.com/1099-4300/22/10/1100.

— (2022). „The Many Faces of Information Geometry.“ en. In: *Notices of the American Mathematical Society* 69.01, p. 1. ISSN: 0002-9920, 1088-9477. DOI: 10.1090/noti2403.

Oord, Aaron Van den, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. (2016). „Conditional image generation with pixelcnn decoders.“ In: *Advances in neural information processing systems* 29.

Oord, Aaron van den, Oriol Vinyals, and koray kavukcuoglu (2017). „Neural Discrete Representation Learning.“ In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2017/hash/7a98af17e63a0ac09ce2e96d03992fbc-Abstract.html.

Oord, Aäron van den, Nal Kalchbrenner, and Koray Kavukcuoglu (2016). „Pixel Recurrent Neural Networks.“ In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, pp. 1747–1756. URL: https://proceedings.mlr.press/v48/oord16.html.

Paszke, Adam et al. (2019). „PyTorch: An Imperative Style, High-Performance Deep Learning Library.“ In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

Pedregosa, F. et al. (2011). „Scikit-learn: Machine Learning in Python.“ In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Perez-Liebana, Diego, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius, and Simon M. Lucas (2019a). „General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Genera-

tion Algorithms.“ In: *IEEE Transactions on Games* 11.3. arXiv:1802.10363 [cs], pp. 195–214. DOI: 10.1109/TG.2019.2901021.

Perez-Liebana, Diego, Simon M. Lucas, Raluca D. Gaina, Julian Togelius, Ahmed Khalifa, and Jialin Liu (2019b). *General Video Game Artificial Intelligence*. Vol. 3. 2. https://gaigresearch.github.io/gvgaibook/. Morgan & Claypool Publishers, pp. 1–191.

Perez, Diego, Spyridon Samothrakis, Simon Lucas, and Philipp Rohlfshagen (2013). „Rolling horizon evolution versus tree search for navigation in single-player real-time games.“ en. In: *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO '13*. Amsterdam, The Netherlands: ACM Press, p. 351. ISBN: 978-1-4503-1963-8. DOI: 10.1145/2463372.2463413. URL: http://dl.acm.org/citation.cfm?doid=2463372.2463413.

Rajabi, M, M Ashtiani, B Minaei-Bidgoli, and O Davoodi (2021). „A dynamic balanced level generator for video games based on deep convolutional generative adversarial networks.“ en. In: *Scientia Iranica*, p. 18.

Rasmussen, Carl Edward and Christopher K. I. Williams (2006). *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, pp. I–XVIII, 1–248. ISBN: 026218253X.

Rezende, Danilo Jimenez, Shakir Mohamed, and Daan Wierstra (2014). „Stochastic Backpropagation and Approximate Inference in Deep Generative Models.“ In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, pp. 1278–1286. URL: https://proceedings.mlr.press/v32/rezende14.html.

Riihimäki, Jaakko and Aki Vehtari (2010). „Gaussian processes with monotonicity information.“ en. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 645–652. URL: https://proceedings.mlr.press/v9/riihimaki10a.html.

Risi, Sebastian and Julian Togelius (2020). „Increasing generality in machine learning through procedural content generation.“ In: *Nature Machine Intelligence* 2.8, 428–436. ISSN: 2522-5839. DOI: 10.1038/s42256-020-0208-z.

Rodriguez Torrado, Ruben, Ahmed Khalifa, Michael Cerny Green, Niels Justesen, Sebastian Risi, and Julian Togelius (2020). „Bootstrapping Conditional GANs for Video Game Level Generation.“ en. In: *2020 IEEE Conference on Games (CoG)*. Osaka, Japan: IEEE, 41–48. ISBN: 978-1-72814-533-4. DOI: 10.1109/CoG47356.2020.9231576. URL: https://ieeexplore.ieee.org/document/9231576/.

Rumelhart, David E. and James L. McClelland (1987). „Learning Internal Representations by Error Propagation.“ In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pp. 318–362.

Rybkin, Oleh, Kostas Daniilidis, and Sergey Levine (2021). „Simple and Effective VAE Training with Calibrated Decoders.“ en. In: *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 9179–9189. URL: https://proceedings.mlr.press/v139/rybkin21a.html.

Salimans, Tim, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever (2017). „Evolution Strategies as a Scalable Alternative to Reinforcement Learning.“ In: arXiv:1703.03864. arXiv:1703.03864 [cs, stat]. URL: http://arxiv.org/abs/1703.03864.

Saravanan, Akash and Matthew Guzdial (2022). „Pixel VQ-VAEs for Improved Pixel Art Representation.“ In: *Experimental AI in Games (EXAG), AIIDE 2022 Workshop*.

Sarkar, Anurag and Seth Cooper (2018). „Blending Levels from Different Games using LSTMs.“ In: *Proceedings of the Experimental AI in Games (EXAG) Workshop at AIIDE*.

— (2020a). „Sequential Segment-based Level Generation and Blending using Variational Autoencoders.“ In: *Proceedings of the 11th Workshop on Procedural Content Generation in Games*.

— (2020b). „Towards Game Design via Creative Machine Learning (GDCML).“ In: *2020 IEEE Conference on Games (CoG)*, 744–751. DOI: 10.1109/CoG47356.2020.9231927.

— (2021a). „Dungeon and Platformer Level Blending and Generation using Conditional VAEs.“ In: *Proceedings of the IEEE Conference on Games (CoG)*.

— (2021b). „Generating and Blending Game Levels via Quality-Diversity in the Latent Space of a Variational Autoencoder.“ In: *Proceedings of the Foundations of Digital Games*.

Sarkar, Anurag, Zhihan Yang, and Seth Cooper (2019). „Controllable Level Blending between Games using Variational Autoencoders.“ In: *Proceedings of the EXAG Workshop at AIIDE*.

Schaul, Tom (2013). „A video game description language for model-based or interactive learning.“ en. In: *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. Niagara Falls, ON, Canada: IEEE, pp. 1–8. ISBN: 978-1-4673-5311-3. DOI: 10.1109/CIG.2013.6633610. URL: http://ieeexplore.ieee.org/document/6633610/.

Schmidhuber, Juergen (2015). „Deep Learning in Neural Networks: An Overview.“ In: *Neural Networks* 61. arXiv:1404.7828 [cs], 85–117. ISSN: 08936080. DOI: 10.1016/j.neunet.2014.09.003.

Schrum, Jacob, Vanessa Volz, and Sebastian Risi (2020). „CPPN2GAN: Combining Compositional Pattern Producing Networks and GANs for Large-Scale Pattern Generation.“ In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO '20. Cancún, Mexico: Association for Computing Machinery, 139–147. ISBN: 9781450371285. DOI: 10.1145/3377930.3389822. URL: https://doi.org/10.1145/3377930.3389822.

Schrum, Jacob, Jake Gutierrez, Vanessa Volz, Jialin Liu, Simon Lucas, and Sebastian Risi (2020). „Interactive evolution and exploration within la-

tent level-design space of generative adversarial networks." en. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. Cancún Mexico: ACM, 148–156. ISBN: 978-1-4503-7128-5. DOI: 10.1145/3377930.3389821. URL: https://dl.acm.org/doi/10.1145/3377930.3389821.

Settles, Burr (2009). *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison. URL: http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf.

Sha, Lingdao, Souju He, Junping Wang, Jiajian Yang, Yuan Gao, Yidan Zhang, and Xinrui Yu (2010). „Creating appropriate challenge level game opponent by the use of dynamic difficulty adjustment." In: *2010 Sixth International Conference on Natural Computation*. Vol. 8, 3897–3901. DOI: 10.1109/ICNC.2010.5584744.

Shahriari, Bobak, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas (2016). „Taking the Human Out of the Loop: A Review of Bayesian Optimization." In: *Proceedings of the IEEE* 104.1, pp. 148–175. DOI: 10.1109/JPROC.2015.2494218.

Shaker, Mohammed, Noor Shaker, and Julian Togelius (2013). „Evolving Playable Content for Cut the Rope through a Simulation-Based Approach." en. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 9.1, 72–78. ISSN: 2334-0924, 2326-909X. DOI: 10.1609/aiide.v9i1.12690.

Shaker, Noor, Julian Togelius, and Mark J. Nelson (2016). *Procedural Content Generation in Games*. en. Computational Synthesis and Creative Systems. Cham: Springer International Publishing. ISBN: 978-3-319-42714-0. DOI: 10.1007/978-3-319-42716-4. URL: http://link.springer.com/10.1007/978-3-319-42716-4.

Shi, Peizhi and Ke Chen (2017). „Learning Constructive Primitives for Real-Time Dynamic Difficulty Adjustment in Super Mario Bros." en. In: *IEEE Transactions on Games* 10.2, 155–169. ISSN: 2475-1502, 2475-1510. DOI: 10.1109/TCIAIG.2017.2740210.

Shu, Tianye, Jialin Liu, and Georgios N. Yannakakis (2021). „Experience-Driven PCG via Reinforcement Learning: A Super Mario Bros Study." In: *2021 IEEE Conference on Games (CoG)*, 1–9. DOI: 10.1109/CoG52621.2021.9619124.

Skafte, Nicki, Martin Jørgensen, and Søren Hauberg (2019). „Reliable training and estimation of variance networks." In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc. URL: https://papers.nips.cc/paper/2019/hash/07211688a0869d995947a8fb11b215d6-Abstract.html.

Smith, Adam M. and Michael Mateas (2011). „Answer Set Programming for Procedural Content Generation: A Design Space Approach." en. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3, 187–200. ISSN: 1943-068X, 1943-0698. DOI: 10.1109/TCIAIG.2011.2158545.

Smith, Gillian, Mike Treanor, Jim Whitehead, and Michael Mateas (2009). „Rhythm-Based Level Generation for 2D Platformers.“ In: *Proceedings of the 4th International Conference on Foundations of Digital Games*. FDG '09. Orlando, Florida: Association for Computing Machinery, 175–182. ISBN: 9781605584379. DOI: 10.1145/1536513.1536548. URL: https://doi.org/10.1145/1536513.1536548.

Snodgrass, Sam and Santiago Ontañón (2016). „Controllable Procedural Content Generation via Constrained Multi-Dimensional Markov Chain Sampling.“ en. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, p. 7.

Snodgrass, Sam and Anurag Sarkar (2020). „Multi-Domain Level Generation and Blending with Sketches via Example-Driven BSP and Variational Autoencoders.“ en. In: *International Conference on the Foundations of Digital Games*. Bugibba Malta: ACM, 1–11. ISBN: 978-1-4503-8807-8. DOI: 10.1145/3402942.3402948. URL: https://dl.acm.org/doi/10.1145/3402942.3402948.

Spronck, Pieter (2005). „Adaptive Game AI.“ PhD thesis. Maastricht, The Netherlands: Maastricht University Press.

Spronck, Pieter, Ida Sprinkhuizen-Kuyper, and Eric Postma (2004). „Online Adaptation of Game Opponent AI with Dynamic Scripting.“ In: *International Journal of Intelligent Games and Simulation* 3.1. Ed. by Neville Gough and Qasim Mehdi, pp. 45–53. ISSN: 1477-2043.

Stanton, Samuel, Wesley Maddox, Nate Gruver, Phillip Maffettone, Emily Delaney, Peyton Greenside, and Andrew Gordon Wilson (2022). „Accelerating Bayesian Optimization for Biological Sequence Design with Denoising Autoencoders.“ In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, pp. 20459–20478. URL: https://proceedings.mlr.press/v162/stanton22a.html.

Sudhakaran, Shyam, Miguel González-Duque, Claire Glanois, Matthias Freiberger, Elias Najarro, and Sebastian Risi (2023). „MarioGPT: Open-Ended Text2Level Generation through Large Language Models.“ In: arXiv:2302.05981. arXiv:2302.05981 [cs]. URL: http://arxiv.org/abs/2302.05981.

Summerville, Adam J and Michael Mateas (2016). „Super Mario as a String: Platformer Level Generation Via LSTMs.“ en. In: *Proceedings of 1st International Joint Conference of DiGRA and FDG*.

Summerville, Adam James, Sam Snodgrass, Michael Mateas, and Santiago Onta n'on Villar (2016). „The VGLC: The Video Game Level Corpus.“ In: *Proceedings of the 7th Workshop on Procedural Content Generation*.

Summerville, Adam, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgard, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius (2018). „Procedural Content Generation via Machine Learning (PCGML).“ In: *IEEE Transactions on Games* 10.3, pp. 257–270. DOI: 10.1109/tg.2018.2846639.

Tanabe, Takumi, Kazuto Fukuchi, Jun Sakuma, and Youhei Akimoto (2021). „Level generation for angry birds with sequential VAE and latent variable evolution.“ en. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Lille France: ACM, 1052–1060. ISBN: 978-1-4503-8350-9. DOI: 10.1145/3449639.3459290. URL: https://dl.acm.org/doi/10.1145/3449639.3459290.

Todd, Graham, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius (2023). „Level Generation Through Large Language Models.“ In: arXiv:2302.05817. arXiv:2302.05817 [cs]. URL: http://arxiv.org/abs/2302.05817.

Togelius, Julian, Sergey Karakovskiy, and Robin Baumgarten (2010). „The 2009 Mario AI Competition.“ In: *IEEE Congress on Evolutionary Computation*, pp. 1–8. DOI: 10.1109/CEC.2010.5586133.

Tomczak, Jakub M. (2022). *Deep Generative Modeling*. en. Cham: Springer International Publishing. ISBN: 978-3-030-93157-5. DOI: 10.1007/978-3-030-93158-2. URL: https://link.springer.com/10.1007/978-3-030-93158-2.

Tournier, Maxime, Xiaomao Wu, Nicolas Courty, Elise Arnaud, and Lionel Reveret (2009). „Motion compression using principal geodesics analysis.“ In: *Computer Graphics Forum*. Vol. 28. 2. Wiley Online Library, pp. 355–364.

Valve Corporation (1998). *Half-Life*. PC. Accessed: 28-03-2023. URL: https://www.igdb.com/games/half-life/presskit.

Volz, Vanessa, Simon M. Lucas, Jacob Schrum, Adam Smith, Jialin Liu, and Sebastian Risi (2018). „Evolving Mario levels in the latent space of a deep convolutional generative adversarial network.“ In: *GECCO 2018 - Proceedings of the 2018 Genetic and Evolutionary Computation Conference*, 221–228. ISSN: 9781450356183. DOI: 10.1145/3205455.3205517.

Wierstra, Daan, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber (2014). „Natural evolution strategies.“ In: *Journal of Machine Learning Research* 15.27. Citation Key: JMLR:v15:wierstra14a, pp. 949–980.

Yannakakis, Georgios N. and Julian Togelius (2011). „Experience-Driven Procedural Content Generation.“ In: *IEEE Transactions on Affective Computing* 2.3, 147–161. ISSN: 1949-3045. DOI: 10.1109/T-AFFC.2011.6.

Zakaria, Yahia, Magda Fayek, and Mayada Hadhoud (2023). „Procedural Level Generation for Sokoban via Deep Learning: An Experimental Study.“ In: *IEEE Transactions on Games* 15.1, 108–120. ISSN: 2475-1510. DOI: 10.1109/TG.2022.3175795.

Zhang, Hejia, Matthew Fontaine, Amy Hoover, Julian Togelius, Bistra Dilkina, and Stefanos Nikolaidis (2020). „Video Game Level Repair via Mixed Integer Linear Programming.“ en. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 16.1, 151–158. ISSN: 2334-0924, 2326-909X. DOI: 10.1609/aiide.v16i1.7424.

Zohaib, Mohammad (2018). „Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review.“ en. In: *Advances in Human-Computer Interaction* 2018, 1–12. ISSN: 1687-5893, 1687-5907. DOI: 10.1155/2018/5681652.