



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Empowering Game Developers through means of Game Software Engineering techniques

Diciembre de 2025

Autor: Jorge Chueca Cerdán

Director: Dr. Óscar Pastor López
Dr. Jaime Font Burdeus

Dedication

A Teresa, que sin esfuerzo iluminas.

A M^a Antonia, José Manuel y Javier, que de mayor quiero ser como vosotros.

A Enrique, Javier, Sergio, Cubero y Pablo, que sigamos hacia arriba y adelante.

A todos, no hay suficiente sitio en este libro para estimar todo vuestro apoyo.

*I knew who I was this morning,
but I've changed a few times since then.*

Lewis Carroll

Acknowledgments

First of all, I want to thank my directors, Dr. Jaime Font and Dr. Óscar Pastor, for their much-needed guidance.

I also want to thank all the SVIT Research Group, who are my role models of hard work, generosity, and selflessness.

And thanks to all my family and friends, you were there even if I was not.

Abstract

Video games are unique software products that have many differences from traditional software. These pieces of software consist of big projects created by heterogeneous teams with the objective of making an emotionally charged experience. To this end, the video game industry needs to provide the player with large amounts of artistic content.

We tackle the content creation and variability management challenges through software engineering techniques. Through industry-scale empirical evaluations, we demonstrate the efficacy and aptness of the proposed approaches: Software Product Lines that help create and manage video game elements, a Search-Based algorithm that generates software models in a Mixed-Initiative fashion between human and machine, and a Phylogenetic strategy towards content creation that unveils latent content within product families.

These techniques needed to be adapted to the particular requirements of the video game craft. Always putting the humans and their creative process at the center of our research.

Resumen

Los videojuegos son productos de software únicos que presentan muchas diferencias respecto al software tradicional. Estas piezas de software consisten en grandes proyectos creados por equipos heterogéneos con el objetivo de realizar una experiencia cargada de emociones. Para ello, la industria del videojuego necesita proporcionar al jugador grandes cantidades de contenido artístico.

Abordamos los retos de la creación de contenidos y la gestión de la variabilidad mediante técnicas de ingeniería de software. Mediante evaluaciones empíricas a escala industrial, demostramos la eficacia y adecuación de los planteamientos propuestos: Líneas de Productos Software que ayudan a crear y gestionar la variabilidad de los elementos de videojuego, un algoritmo Basado en Búsqueda que genera modelos de software de mediante Iniciativa Mixta entre humano y máquina, y una estrategia Filogenética acerca de la creación de contenido que revela contenido latente dentro de las familias de productos.

Estas técnicas necesitan adaptación a las exigencias particulares del sector del videojuego. Siempre poniendo al ser humano y su proceso creativo en el centro de nuestra investigación.

Resum

Els videojocs són productes de software únics que presenten moltes diferències respecte al software tradicional. Aquestes peces de software consisteixen en grans projectes creats per equips heterogenis amb l'objectiu de realitzar una experiència carregada d'emocions. Per a això, la indústria del videojoc necessita proporcionar al jugador grans quantitats de contingut artístic.

Abordem els reptes de la creació de continguts i la gestió de la variabilitat mitjançant tècniques d'enginyeria del software. Mitjançant avaluacions empíriques a escala industrial, demostrem l'eficàcia i l'adequació dels plantejaments proposats: Línies de Productes Software que ajuden a crear i gestionar la variabilitat dels elements dels videojocs, un algorisme Basat en Cerca que genera models software mitjançant Iniciativa Mixta entre humà i màquina, i una estratègia Filogenètica sobre la creació de continguts que revela contingut latent dins de les famílies de productes.

Aquestes tècniques necessiten una adaptació a les exigències particulars del sector del videojoc. Sempre posant l'ésser humà i el seu procés creatiu al centre de la nostra investigació.

Contents

Dedication	iii
Acknowledgments	v
Abstract - Resumen - Resum	vii
Contents	xiii
I Introduction	1
II Compendium of Articles	23
1 Comparing Software Product Lines and Clone and Own for Game Software Engineering under two paradigms	25
2 Search-based Co-creation of Software Models: The case of particle systems for video games	69

3 Leveraging Phylogenetics in Software Product Families: The Case of Latent Content Generation in Video Games	109
III Discussion	143
IV Conclusions	153

Part I

Introduction

Introduction

Video games are a multifaceted field. One key artifact is the main bottleneck in their development. Content. Through Game Software Engineering, we aim to improve content creation and management challenges.

Context

There is no piece of literature regarding video games that does not start by acknowledging the rapid growth, current size, or complexity of the video game industry. All of them do that for good reasons, and this text is no less.

The video game industry's global revenue is higher than that of the music and movie industries *combined* [18]. This is the go-to fact about this industry when trying to express the importance and influence of video games. However, this fact only speaks of the economic results of video game development. While the economic part is important, video games have other particularities and facets that make this field unique to Software Engineering researchers.

Game Software Engineering (GSE)

The financial success is mirrored by the increasing complexity and scale of video game software, leading to the expansion of development teams. In 1980, *Namco* released the arcade game *Pac-Man* that entertained families worldwide. This game was developed over a one-year and five-month period by four developers [14, 16], including a music composer. Today, the narrative towards playing video games has changed. With more than a 40% of the global population as players, video games are not mere entertainment anymore [1, 8, 6], now; they are considered an art form; they are used to teach [6, 20], rehabilitate the sick [9, 10], perform digital music concerts [19], and more. It has been less than 40 years since video games were considered toys, to experiences that are large, complex worlds with such historical fidelity that they are used in education and museums. This is the case of *Assassin's Creed: Origins*, a video game developed by 4,579 people during approximately four years [13]. This development included a myriad of roles: programmers, designers, 2D and 3D artists, animators, musicians, producers, actors, and even historians were employed.

This accelerated growth has led to many super-productions and the concept of Games as a Service (GaaS). These video games have no proper “end”, and they are updated periodically with new content. Nowadays, video games are being marketed with narratives that only grow players' expectations. Many buzzwords are used, such as: *open-world*, *endless replayability*, or *hundreds of hours of content*. Players naturally respond by demanding high-quality games with impressive features and extensive content with seamless performance. They expect video games to improve year after year with bigger and better

games. This situation affects teams of all sizes, from highly realistic super-productions to small independent developers.

One particularity that arises from the complexity of video games is Game engines. They provide core functions like graphics rendering, audio, physics, and AI, and can include scripting modules, assets, and level editors. Game engines also often add scripting capabilities to streamline the development workflow. One of the main reasons developers build game engines is to “better control the environment and source code” [18]. In traditional software development, a developer often uses a separate IDE to write code that interacts with one or more software frameworks; in game development, the game engine often serves as both the underlying framework and the primary development environment. Only the artists and designers use other tools and frameworks in the development.

The combination of intricate systems, big heterogeneous teams, subjective requirements, and creating and managing large amounts of content at high refresh rates makes video games arguably the most complex software product being developed today. Such is the case that the specific needs of video game software have driven technological advancements. Notably, the development of modern GPUs was driven by the demand for high visual fidelity in games [7]. We turn to Software Engineering (SE) to address the challenges that video games entail.

This branch of SE is **Game Software Engineering (GSE)** that has emerged as a distinct scientific domain, shaped by the unique demands of video game development [5]. Unlike traditional software, video games combine complex technical systems with artistic and creative elements, aiming not just for functionality but for entertainment and engagement. Development teams are inherently multidisciplinary, involving roles from art, music, animation, and marketing alongside software engineers. The process is highly iterative, with design and implementation often overlapping, leading to a further differentiation of GSE from conventional SE practices.

Additional challenges in GSE include managing vast numbers of digital assets to meet high user expectations, and conducting testing that goes beyond technical validation to include aspects such as aesthetics, game mechanics, and balance. The development also involves the coordination of multiple complex subsystems such as rendering, physics, audio, and AI, often through the use of specialized game engines and scripting languages. Furthermore, the need to publish across a wide range of platforms introduces significant technical complexity. Together, these factors define GSE as a unique and rapidly evolving

discipline that calls for dedicated research into its specific methods, tools, and challenges.

Finally, another distinctive aspect of GSE is its emphasis on subjective requirements, most notably the “fun” factor, which is crucial for player engagement and enjoyment. However, defining and achieving “fun” is inherently subjective, varying greatly depending on personal preferences, cultural background, and individual expectations. This subjectivity presents a challenge for software engineers who are accustomed to dealing with more objective and measurable requirements. Unlike traditional software, where requirements are often tied to real-world functionalities and performance metrics, video games must cater to a diverse range of emotional responses and player motivations. The need to balance technical functionality with artistic expression and subjective player satisfaction makes video game development a unique and challenging domain within SE.

Objectives

As video games have such a particular software identity, traditional SE techniques must be adapted or new ones created. Not only can GSE improve video game development, but video games can also serve as a powerful tool to improve SE as a whole by providing unique scenarios and case studies. The main motivation for this thesis is to help video game developers through software engineering. However, we want to specify that the video game domain and SE can cross-fertilize each other.

Content is the main problem that needs to be tackled in video game development. Our main objective in this thesis is to find SE techniques that can help developers manage large amounts of content variability and create new content in an optimal way. For this purpose, we take into account the industrial context and practitioners’ expertise. By making this thesis industry-aware, we put the focus of our research on the developers. To evaluate our different approaches, we use interviews with professionals, industrial use cases, or professional craft made specifically for research.

The demand for large volumes of content in games like Kromaia often turns content creation into a significant bottleneck in the development process. Video game content encompasses a wide variety of interdisciplinary elements, each of which contributes to the interactive experience. These components include executable code, texture maps, 3D meshes, non-player characters (NPCs), level designs, musical scores, narrative structures, user interface, and more. Inte-

grating all these elements requires a coordinated effort between all the multifaceted roles of GSE: software engineering, visual arts, narrative design, and audio production. Two main challenges arise from this situation: how to create such a quantity of content in a correct and efficient manner, and how to manage all that content once it is created. From these objectives, we formulate the following Research Questions:

RQ1. Can SE techniques be used to improve **content generation** in video games? If so, how do these techniques need to be adapted to tackle the video game domain’s particularities?

RQ2. Can SE techniques be used to improve **variability management** in video games? If so, how do these techniques need to be adapted to tackle the video game domain’s particularities?

Background

SE has a myriad of techniques designed to improve the efficiency, reliability, and maintainability of software systems. In this section, we introduce an overview of the key methodologies relevant to content creation and variability management in GSE, and thus, relevant to this thesis.

Software Product Line (SPL)

One approach that is key to the development of this thesis is Software Product Line (SPL) engineering. A Software Product Line (SPL) is a set of related software-intensive systems that share a common, managed set of features designed to meet specific requirements. It is based on two key processes: domain engineering, which defines the variability across the product line and creates reusable artefacts; and application engineering, which uses these artifacts to build specific products by selecting and configuring predefined variations. [17]. It is unavoidable not to think of SPL for content creation and variability management because of its focus on systematic reuse.

Model-Driven Engineering (MDE)

SPLs can also be used in an MDE context. While SPLs manage variability through domain models that define common and variable features, MDE provides the tools to transform these models into executable code. This synergy supports SPL’s goals of reuse, cost reduction, and faster development.

Model Driven Engineering (MDE) is another of the fundamental techniques we use in this thesis. MDE is an approach to software development that focuses on using abstract models to create software. Instead of creating applications by coding line-by-line. Developers can use MDE to design high-level representations of a system’s structure, behavior, and logic. These models generate executable code and other software artifacts because the model is the code. The main purpose of MDE is to separate the conceptual design of the system from the underlying technical implementation, which helps with complexity management, improves maintainability, and enables cross-platform development. This abstraction layer allows developers to focus more on the “what” than the “how” of software design [15].

One industrial success case of MDE can be seen in the video game industry, this is the Blueprint system in Unreal Engine. Blueprints provide a visual scripting environment that allows developers and designers to build game systems, character behaviors, and rendering without writing traditional code. By dragging and connecting nodes that represent code users can create a model of the game logic. The engine then compiles these models into executable code. By lowering the barrier for non-programmers to code, this speeds up prototyping and iteration. MDE is a key ingredient for video game development nowadays.

Search-Based Software Engineering (SBSE)

Finally, we want to manage and create large ammounts of content that often rely on subjective requirements. In order to create algorithmically the creative content that populates video games we recur to Search-based software engineering (SBSE). SBSE is an SE approach that leverages computer algorithms to discover optimal (or near-optimal) solutions to SE tasks. SBSE helps with the automatic discovery of heuristic-based solutions that are too difficult to discover manually. SBSE excels in contexts where several conflicting objectives must be balanced, such as maximizing performance and minimizing resource usage [11].

Throughout this thesis, we use a common algorithm in SBSE, the Genetic Algorithm (GA), and variations from it. GAs are inspired by the principles of natural selection and evolution. In a GA, a population of candidate solutions is generated, each representing a potential answer to the problem. These solutions are evaluated using a fitness function that measures their quality. The best-performing solutions are selected to reproduce, combine, and mutate their features to create a new generation of candidates. Over successive generations, the population ideally evolves toward better solutions. This organic nature fits perfectly for the task at hand of creating subjective solutions (content).

Case Studies

As this thesis has an important focus on the industrial aspect of research we evaluated the different approaches on industrial case studies. Kromaia is the most representative industrial artifact analyzed in this thesis and serves as a key example in our study of GSE. Developed by Kraken Empire, it is a commercial 3D space shooter available on PC and console, known for its distinctive 360-degree movement and shooting system. Players pilot a spaceship through environments filled with enemies, obstacles, and collectibles, with each level ending in a boss encounter. Technically, the game is built on a proprietary engine and incorporates external libraries for rendering and for physics simulation.

Research Methodology

The methodology we followed to develop this thesis is the one proposed by R. Wieringa [21]. The research originates from the need to improve a particular aspect of game development. In our case, the bottleneck of content generation that video games suffer and managing all those assets. To address this need, the research focuses on developing novel approaches that can be applied in the context of video game development with the proper evaluation of the results. The ultimate goal is to provide game developers with techniques that facilitate the creation of high-quality games.

Similar to video game development, the Research Cycle methodology is iterative, as we can see in Figure 1. The five different phases followed are:

1. Problem Investigation: Identifying specific challenges or problems in video game development. This could be related to any stage of the game de-

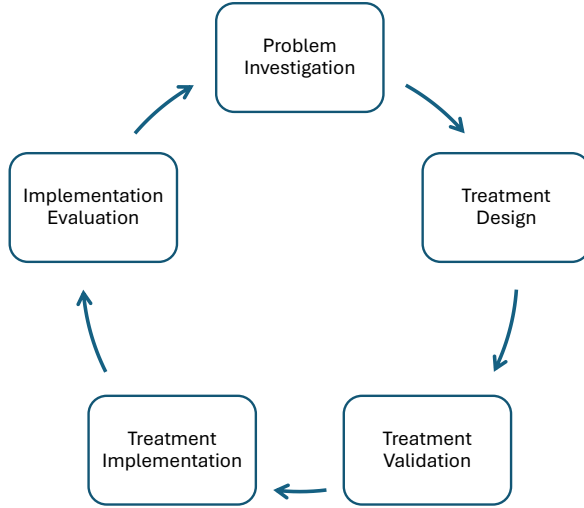


Figure 1: The research cycle.

velopment pipeline, from game design and prototyping to testing and deployment.

2. **Treatment Design:** Developing novel approaches to address the identified problem. These solutions are tailored towards answering the research questions.
3. **Treatment Validation:** Identifying and solving the possible threats to the validity of the approach. For this purpose, we use widely accepted evaluation metrics and perform empirical evaluations. Additionally, as this thesis aims to improve the video game industry, we validated the designs through industrial case studies, as well as practitioners and experts.
4. **Treatment Implementation:** Developing the approaches and applying them to the industrial cases.
5. **Implementation Evaluation:** Conducting the empirical evaluation and analysis of the results. This analysis is later discussed with the researcher’s opinion, answering the research questions.

The result of a single cycle is the implemented approach and the manuscript detailing the research conducted. As the process is iterative, the new knowledge is built on top of the previous findings.

Thesis Overview

In this section, we chronologically explain the work that results in this thesis. Table 1 portrays the summary of the work included as a compendium of publications in this book. The research work started with an extensive analysis of GSE at an industrial scale. For this, we have developed a Systematic Literature Review. The paper titled “*The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games*” [5] resulted in the analysis of 98 papers on GSE industry-scale research, earmarking the particularities, trends, and challenges of GSE. It was published in the journal Information and Software Technology (IST). There are many results from this work signaling the advancement of GSE research in consonance with the growth of the video game industry. Related to this thesis, one key finding from this work is that Software Product Lines (SPLs) can ease the challenges of software variability in the context of the subjectivity of video games.

Studies	SLR GSE	Comparing SPL vs CaO	Software Model Co-Creation	Latent Content Generation
Technique		SPL	GA & MI	SPL & Phylogenetics
Objective	Analyze SotA	Generate Content & Variability Management	Generate Content	Generate Content & Variability Management
Artifact	Industry GSE Research	Software Models & Code	Software Models	Software Models
Tackled Video Game Characteristics	Identified all Video Game Characteristics	Multidisciplinary Developers with non-engineering background		
		Manage large amounts of Content to meet customers' high expectations		
			Subjective Requirements, such as “fun”	
Publications	IST'24 [5]	JSS'23 [2]	IST'24 [4]	SPLC'24 [3]
Presentations		CEV'23		SPLC'24
Main Takeaways	GSE is an independent research field	- SPLs improve development - Devs fear the loss of agency	Mixed-Initiative is optimal for creative work	Phylogenetics helps meet subjective requirements
Empirical Evaluation		Experiment		Simulated player

Table 1: Summary of the publications included in this thesis.

The work for this thesis is comprised of four studies. Table 1 shows them in the first row: *SLR GSE*, a Systematic Literature Review of industry-scale GSE research from 2009 to 2021; *Comparing SPL vs CaO*, an empirical analysis comparing SPL vs CaO as development approaches for video games; *Software Model Co-Creation*, a variation of a GA that creates creative software models in a Mixed-Initiative fashion between human and machine; and *Latent Content Generation*, a novel approach to generate content through variability analysis using phylogenetics. In the Technique row is the main SE technique used. In the Objective row is the main objective that the study tries to achieve. In the Artifact row is the paradigm in which each study is developed, mainly

MDE. The Tackled Video Game Characteristic row portrays the main facet of GSE to which we are adapting the technique. The Publications row shows the publication venue for each study with its corresponding citation. The Presentations row shows the name of each conference where each study has been presented. The Main Takeaways row shows an extremely short summarization of the conclusions of each study. Finally, the Empirical Evaluation row shows the evaluation methods for our approaches.

First, we analyzed the state of the art for all GSE research at industry-scale. From this work, we recognized GSE as a fertile ground for research, either by improving SE techniques with video game case studies or by improving game development via adapted SE techniques. We followed the best practices to perform a Systematic Literature Review (SLR) by Kitchenham et al. [12]. The paper, later published in the journal Information and Software Technology (IST), is titled: *“The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games”*. In this paper, we identify the main characteristics that differentiate traditional SE from GSE, and we earmark 13 different future directions of this research field.

With this knowledge, we decided to evaluate empirically the feasibility of SPLs in video games. We performed an experiment involving 79 subjects in different paradigms used to develop video games, comparing the SPL and Clone and Own (CaO) techniques. The manuscript that stems from this work is titled *“Comparing software product lines and Clone and Own for game software engineering under two paradigms: Model-driven development and code-driven development”* [2], published in the journal Journal of Systems and Software (JSS) and later presented at the conference II Congreso Español de Videojuegos (CEV). Among the different results, one is the most intriguing. While most of the results are in favor of SPLs over CaO, the intention of use was clearly in favor of CaO. Game developers argue that they fear the loss of control over their work using SPLs.

Building on the knowledge gained, we marked the next steps in the research involving humans in the algorithmic process that is often fully in the hands of the computer. With this, game developers will have more agency over their work. This next research resulted in an approach that generates video game software models that have a creative nature. In each cycle of the algorithm, humans are presented with a diversity of models, and they select the most fitting ones for their subjective purposes. In a Mixed-Initiative fashion, the computer runs a genetic algorithm guided by the selection of humans. This work is explained in the paper titled *“Search-based co-creation of software mod-*

els: The case of particle systems for video games” [4] published in the journal Information and Software Technology (IST). The approach was evaluated with an experiment involving a game development team with different roles tasked to create particle system effects. Visual effects such as particle systems are a mix of technical and artistic knowledge. They are present in almost every video game. This approach makes it possible for non-experts to make creative models that are of similar quality to those made by an expert in a fraction of the time.

Finally, for the last iteration of the research cycle in this thesis, we developed an approach related to SPLs, again keeping in mind the previous findings. We employed a Phylogenetic approach to video game content variability. This work titled *“Leveraging Phylogenetics in Software Product Families: The Case of Latent Content Generation in Video Games”* [3] was published and presented at the conference 28TH ACM International Systems and Software Product Line Conference (SPLC). This approach is twofold. First, the phylogenetic tree helps developers understand the great variability that video game content entails. Second, with the phylogenetic relationships among individuals, we can create new content that is ensured to fit more into the subjective requirements of a particular video game.

Research support

In the development of this thesis, we have received the support of multiple Funding Research Projects, Excellence Networks, and Industrial partners. Table 2 presents all the funding sources and entities that supported this research.

The first four rows are the research projects that funded this thesis’s work, in descending order: The VARIATIVA project proposes the application of interactive search on software models as a starting point to establish software product lines. The project explores different possibilities to integrate the human in the search process, with different degrees of involvement (providing input, selecting output, or even totally replacing the fitness function). The Mixed-initiative algorithm presented in the IST’24 [4] paper contributes to the project by empowering the human with two main tools to guide the search algorithm: (1) the human can alter the fitness function at runtime, while the search is being performed, to guide it to what it wants to find; (2) the partial results presented by the search algorithm can be previewed during the search and reordered along different dimensions, which encourages exploration of the areas of the search space where solutions most interesting to the human are

Funded Research Projects	VARIATIVA	<i>Iniciando Líneas de Producto Software Mediante Búsquedas Interactivas Dirigidas por Modelos.</i> [PID2021- 128695OB-I00]
	DOLMEN	<i>Inteligencia computacional para iniciar líneas de producto software de contenido de videojuegos.</i> [SAV-20221011]
	VARNETICA	<i>Reingeniería de Líneas de Producto Software en Modelos Software mediante Filogenética Evolutiva.</i> [CNS2023-145422]
	Gobierno de Aragón	Research group SOFTWARE VARIABILITY FOR INFORMATION TECHNOLOGIES (SVIT). [T61_23R]
Excellence Networks	AI4SE	Research Network on Artificial Intelligence Applied to the Software Development Process. [RED2022-134647-T]
	TASOVA PLUS	Network on new Trends in Software Architecture and Variability. [RED2022- 134337-T]
Industrial Partners	Kraken Empire	Company focused on creating technology using physics, AI, and innovative user experiences.
	Entalto Games	Company focused on multiplatform publishing and advanced software solutions for video games.

Table 2: Funded Research Projects, Excellence Networks, and Industrial Partners

obtained. Providing humans with more tools to control the search facilitates exploration in uncertain scenarios where the search goal is not easy to formalize, such as the case of particle systems presented in the paper. The DOLMEN project seeks to improve content generation in video games through computational intelligence and SPL. The three papers in this thesis contribute to the project by offering new approaches to content generation within GSE. The JSS’23 [2] paper demonstrates the effectiveness of SPLs in generating multifaceted content. The IST’24 [4] paper leverages the Initiative-Mixed algorithm to create creative content. And finally, the SPLC’24 [3] paper provides phylogenetics as a way to create content from variability management. Phylogenetics is the branch of evolutionary biology that studies the evolutionary history of groups of organisms and their relationships through phylogenetic trees. The VARNETICA project proposes to transfer its application to software models, using it to re-engineer software product lines. Our SPLC’24 [3] work contributes to VARNETICA, serving as a proof of concept of how phylogenetics can contribute to managing the existing variability among the products of an SPL and to exploiting that variability to generate new products aligned with the SPL. The operations created to compare the software models of elements of a video game have proven to be useful and will be reused in future work related to this project. The SVIT project for the DGA call 23-25 focuses on the reengineering of software product lines for video games and the use of evolutionary algorithms for their establishment. The JSS’23 [2] work contributes to the project as it has allowed us to analyze the use made of SPLs for the de-

velopment of video games, delving into the reasons why the developer chooses an SPL or not for their video games. In addition, the project also seeks the creation of genetic operations and fitness functions specific to video games, such as the ones we propose in the IST'24 [4] work.

The next two rows are the excellence networks in which some of the work is represented, in descending order: The AI4SE Network of Excellence is dedicated to promote research and collaboration in the use of Artificial Intelligence applied to Software and Systems Engineering. Its goal is to improve and automate processes such as requirements analysis, design, testing, refactoring and software maintenance through advanced AI techniques, fostering cooperation between research groups, scientific dissemination and the development of innovative projects. The TASOVA Plus Network - "New Trends in Software Architecture and Variability" is dedicated to fostering national and international collaboration among 13 universities around software architecture innovation and variability management. Its objective is to strengthen synergies, visibility and knowledge transfer in key areas such as adaptive architectures, digital twins and variability modeling, through seminars, workshops among research groups, and workshops in the field of software architecture and variability management.

Finally, the last two rows are the industrial partners that helped the research with their expertise and case studies, in descending order: Kraken Empire is an independent game development studio founded in 2010 and based in Zaragoza, Aragon. They are dedicated to creating original games that combine creativity with engineering, highlighting the use of physics simulation, artificial intelligence and real-time systems. Throughout their trajectory they have launched titles such as Kromaia, a 3D shooter with total freedom of movement; Hi Score Science, an educational application of scientific questions; and Toy Tactics, an innovative real-time strategy game where the player draws tactical formations. Entalto Games is an independent game development, publishing, and porting studio based in Zaragoza, founded in 2020-2021. Formed by a small team (8 people), they multiplatform development (PC, consoles, mobile, VR) using different game engines. Their first games include NeonHAT (2021), Vampire: The Masquerade - Heartless Lullaby (2021) and Grabitoons! (2022); they have also published and distributed their own and third-party games such as Stickin' the Landing, Manitas Kitchen, or Two Strikes. They also operate as a publisher under the label "Entalto Publishing", offering support in physical and digital release.

Articles Compendium

Multiple articles have been published as the result of the research for this thesis. In chronological order:

1. **Comparing Software Product Lines and Clone and Own for Game Software Engineering under two paradigms** (Journal of Systems and Software) (Chueca, Trasobares, Domingo, Arcega, Cetina, Font, 2020) (**JSS'23**) [2]:

This paper presents the results of two experiments that investigated whether adopting Software Product Lines (SPLs) in Game Software Engineering (GSE) can lead to similar benefits as seen in Classic Software Engineering (CSE). The experiments compared two development approaches - Clone and Own (CaO) and SPLs - under Model-Driven Development (MDD) and Code-Driven Development (CDD) paradigms. 79 subjects with varying experience levels participated in the experiments by developing final bosses for a commercial video game. SPL creation resulted in a significant improvement against CaO in game development. The only metric that did not improve was the intention of use, earmarking the developers' fear of control over their work.

2. **Search-based co-creation of software models: The case of particle systems for video games** (Information and Software Technology) (Chueca, Cetina, Pastor, Font, 2024) (**IST'24**) [4]:

This paper presents a new approach for co-creating video game content using a combination of evolutionary algorithms (specifically, MAP-Elites), software models, and human input. The approach generates a diverse set of software models instead of a single optimal individual determined by the fitness function. This allows human users to guide the selection process based on their creative preferences. An evaluation using Particle Systems (PS) demonstrated that non-technical professionals can generate PS similar to professional human-made PS. This indicates that human-guided evolutionary algorithms can effectively support creative content creation.

3. **Leveraging Phylogenetics in Software Product Families: The Case of Latent Content Generation in Video Games** (28TH ACM International Systems and Software Product Line Conference) (Chueca, Blasco, Cetina, Font, 2024) (**SPLC'24**) [3]:

This paper presents a new approach to generating content for software product families by applying phylogenetics, a technique used in biology to study evolutionary relationships between organisms. The approach involves creating a phylogenetic tree of the existing products in a family to identify potential gaps (ancestors). These ancestors represent latent content that could be reconstructed from the existing products (descendants). The Latent Content Generation (LCG) approach resulted in content fitting more into the game metrics than traditional Procedural Content Generation (PCG).

Thesis Structure

Following the guidelines of a thesis as a compendium of articles by the Universitat Politècnica de València, the structure of the thesis is as follows:

I Introduction: This first part of the thesis introduces the particularities of the video game context, the objectives of the research, the research methodology followed, an overview of the thesis, the list of scientific articles included in the thesis and finally the entities that supported this research.

II Compendium of Articles: The second part of the thesis presents the compendium of scientific articles that result from the research. These articles are the author's versions adapted to the format of this thesis, ordered chronologically.

III Discussion: The third part of the thesis answers the two research questions that guided the research, explaining how GSE can help the creation and management of video game development. This section also looks into the future of the ongoing GSE research.

IV Conclusions: The fourth and final part of the thesis compiles the final thoughts on the thesis work.

Bibliography

- [1] Michiel Buijsman. *Global Games Market Report 2024*. [Online; accessed 17-June-2025]. 2024. URL: <https://newzoo.com/resources/trend-reports/newzoos-global-games-market-report-2024-free-version> (cit. on p. 5).
- [2] Jorge Chueca et al. “Comparing software product lines and Clone and Own for game software engineering under two paradigms: Model-driven development and code-driven development”. In: *Journal of Systems and Software* 205 (2023), p. 111824. ISSN: 0164-1212. DOI: <https://doi.org/10.1016/j.jss.2023.111824> (cit. on pp. 12, 13, 15, 17).
- [3] Jorge Chueca et al. “Leveraging Phylogenetics in Software Product Families: The Case of Latent Content Generation in Video Games”. In: *Proceedings of the 28th ACM International Systems and Software Product Line Conference*. SPLC ’24. Dommeldange, Luxembourg: Association for Computing Machinery, 2024, 113–124. ISBN: 9798400705939. DOI: 10.1145/3646548.3672596 (cit. on pp. 12, 14, 15, 17).
- [4] Jorge Chueca et al. “Search-based co-creation of software models: The case of particle systems for video games”. In: *Information and Software Technology* 171 (2024), p. 107466. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2024.107466> (cit. on pp. 12, 14–17).
- [5] Jorge Chueca et al. “The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games”. In: *Information and Software Technology* 165 (2024), p. 107330. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2023.107330> (cit. on pp. 6, 12).
- [6] J. Clement. *Video gaming worldwide - Statistics & Facts*. Accessed online. June 3, 2024. URL: <https://www.statista.com/topics/1680/gaming/#topic0verview> (visited on 05/19/2025) (cit. on p. 5).
- [7] William J. Dally, Stephen W. Keckler, and David B. Kirk. “Evolution of the Graphics Processing Unit (GPU)”. In: *IEEE Micro* 41.6 (2021), pp. 42–51. DOI: 10.1109/MM.2021.3113475 (cit. on p. 6).
- [8] F. Duarte. *How Many Gamers Are There? (New 2025 Statistics)*. Accessed online. Apr. 24, 2025. URL: <https://explodingtopics.com/blog/number-of-gamers> (visited on 05/19/2025) (cit. on p. 5).

-
- [9] Carla Sílvia Fernandes et al. “Exergames for rehabilitation in stroke survivors: Umbrella review of meta-analyses”. In: *Journal of Stroke and Cerebrovascular Diseases* 34.1 (2025), p. 108161 (cit. on p. 5).
- [10] Lynne V Gauthier et al. “Video game rehabilitation for outpatient stroke (VIGoROUS): a multi-site randomized controlled trial of in-home, self-managed, upper-extremity therapy”. In: *EClinicalMedicine* 43 (2022) (cit. on p. 5).
- [11] Mark Harman and Bryan F Jones. “Search-based software engineering”. In: *Information and software Technology* 43.14 (2001), pp. 833–839 (cit. on p. 9).
- [12] Barbara A. Kitchenham et al. “Preliminary guidelines for empirical research in software engineering”. In: *IEEE Transactions on Software Engineering* 28.8 (Aug. 2002), pp. 721–734. ISSN: 0098-5589 (cit. on p. 13).
- [13] MobyGames. *Assassin’s Creed: Origins. Windows credits (2017)*. <https://www.mobygames.com/game/97558/assassins-creed-origins/credits/windows/> [Accessed: 22/08/2024] (cit. on p. 5).
- [14] MobyGames. *Pac-Man. Arcade credits (1980)*. <https://www.mobygames.com/game/138/pac-man/credits/arcade/> [Accessed: 22/08/2024] (cit. on p. 5).
- [15] Oscar Pastor and Juan Carlos Molina. *Model-driven architecture in practice: a software production environment based on conceptual modeling*. Vol. 1. Springer, 2007 (cit. on p. 9).
- [16] Jamey Pittman. *The Pac-Man Dossier*. https://web.archive.org/web/20200616074003/https://www.gamasutra.com/view/feature/132330/the_pacman_dossier.php [Accessed: 22/08/2024] (cit. on p. 5).
- [17] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Vol. 1. Springer, 2005 (cit. on p. 8).
- [18] Cristiano Politowski et al. “Are game engines software frameworks? a three-perspective study”. In: *Journal of Systems and Software* 171 (2021), p. 110846 (cit. on pp. 5, 6).
- [19] Andrew Smith. *All Fortnite Live Events So Far*. <https://www.gamespot.com/gallery/all-fortnite-live-events-so-far/2900-4115/> [Accessed: 17/10/2024] (cit. on p. 5).

- [20] Maja Videnovik et al. “Game-based learning in computer science education: a scoping literature review”. In: *International Journal of STEM Education* 10.1 (2023), p. 54 (cit. on p. 5).
- [21] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014 (cit. on p. 10).

Part II

Compendium of Articles

Chapter 1

Comparing Software Product Lines and Clone and Own for Game Software Engineering under two paradigms

Game developers often face more challenges when reusing code compared to non-game developers, and Software Product Lines (SPLs) have been successful in addressing this issue. In this study, we compare different approaches to code reuse in Classic Software Engineering (CSE) and Game Software Engineering (GSE) through a commercial video game called Kro-maia. We specifically focus on two development paradigms: Model-Driven Development (MDD) and Code-Driven Development (CDD). We conduct an empirical evaluation where subjects develop game elements using two approaches: Clone and Own (CaO) and SPLs. The results show that game elements developed using SPLs are more correct (over 23%) than those developed with CaO in both MDD and CDD paradigms. In CDD, there are significant improvements in efficiency (51%) and satisfaction (13%) when using SPLs compared to CaO. However, no improvements are observed when working under MDD. The impact of using SPLs or CaO is greater in CDD than in MDD for game developers. Our findings suggest that SPLs in GSE may have a different role compared to their traditional role in CSE. Specifically, SPLs can be valuable in balancing game difficulty or generating new video game content, such as the one present in the bosses of the game.

Author's version of the paper: **Jorge Chueca**, José Ignacio Trasobares, África Domingo, Lorena Arcega, Carlos Cetina, Jaime Font, "Comparing software product lines and Clone and Own for game software engineering under two paradigms: Model-driven development and code-driven development" *Journal of Systems and Software*, vol. 205, p. 111824, Nov. 2023, doi: <https://doi.org/10.1016/j.jss.2023.111824>.

1.1 Introduction

Nowadays, the video game industry is one of the fastest-growing industries in the world. According to a 2019 report [46], the total number of active software developers is 18.9M. The same report indicates that 8.8M of these active developers have worked or are working on video games. This means that almost half of the active developers have been involved in video game development at some point. Video game development has differences from classic software development [26, 36]. Hence, it is important to note that Game Software Engineering (GSE) aligns with Classic Software Engineering (CSE), sharing many similarities rather than significant differences [1]. One difference is that game developers often face more challenges when reusing code compared to non-game developers and do not have standards to avoid technical debt [8, 42].

The majority of video games are developed using game engines, which are development environments that integrate two engines for graphics and physics and a set of tools to accelerate development. The most popular are Unity [48] and Unreal Engine [20], but it is also possible for a studio to make its own engine (e.g., CryEngine [12]). Game engines allow video game developers to create content directly using code (e.g., C++) or software models.

Software models raise the abstraction level using terms that are much closer to the problem domain. The Model-Driven Development (MDD) [44] paradigm focuses on the creation of models that can then be translated into source code. This means that developers can focus on the game content itself, abstracting from the implementation details. In addition, game engines also allow working under the traditional Code-Driven Development (CDD) paradigm. Working at the code level allows developers to have more control over the implementation details when needed.

The use of Software Product Lines (SPLs) has proven to be effective in CSE for developing different types of software at a lower cost, in less time, and with higher quality [41]. That is why there are recent research efforts that propose applying SPLs in the domain of video games [29, 32, 45]. However, recent research [40] has provided evidence that game developers perceive more difficulties than non-game developers when reusing code. Contrary to a systematic development method exists Clone and Own (CaO), it involves the informal ad-hoc practice of extracting reusable artifacts (e.g., code) from the software and duplicating them in other parts of the program [19]. This practice is commonly known as *copy and paste*.

This paper is an extension of our previous work [50] in which we performed an empirical evaluation to compare two development approaches, CaO and an SPL under the MDD paradigm. The evaluation was performed using Kromaia, a commercial video game that other authors have previously analyzed [6, 16, 17, 5]. The 28 participants had to develop two final bosses of Kromaia, one using CaO and another one using a compositional SPL. The results were compared in terms of correctness, efficiency, and user satisfaction. Although the bosses developed with the SPL were more correct than those developed with CaO, the results did not indicate significant changes in efficiency or satisfaction.

The evaluation of the previous work focused only on the MDD paradigm; in this work, we extend the empirical evaluation of our previous work to include a new development paradigm in the analysis, the CDD paradigm. Although the MDD paradigm is being used by video game developers, working at the level of source code is more common in GSE. With this work, we want to determine if the benefits of using an SPL rather than CaO when developing video game content under the MDD paradigm discovered in the previous work still apply under the CDD paradigm. Furthermore, we will analyze the aggregated results from the MDD and CDD paradigms to determine the effect of the paradigm on the results.

To this end, 51 new subjects were recruited to develop two final bosses of Kromaia in C++. Each boss was created following a different development approach: one using CaO, reusing code from existing bosses and libraries, and another using an annotative SPL that derives final bosses in C++ code. Then, the results were analyzed and compared following the same indicators of correctness, efficiency, and user satisfaction employed in our previous work.

The empirical evaluations we perform are presented from the perspective of *controlled experiments* [54] in which humans apply different treatments to one or more objects of study in a laboratory setting. The use of a laboratory

setting allows greater control over the manipulation and measurements of the variables involved in the study in order to test hypotheses or determine causal relationships. We acknowledge that some readers may refer to this study as a *case study*. However, we use the Wohlin et al. terminology for empirical studies [54]. In this context, the term *case study* does not fit our empirical evaluation because although we use a real video game to extract the objects to evaluate the approaches SPL & CaO, our study does not take place in a real-world context but in laboratory settings and the focus of the study is not the video game analysis itself.

Our results suggest that the SPLs obtain better results than CaO, especially under the CDD paradigm. However, the SPL approach in GSE does not have the same effects that have made SPLs attractive to CSE. In light of our results, we may need to rethink the role of SPLs for GSE, treating GSE as a different domain than CSE when dealing with SPLs. Our work suggests new research directions for SPL in GSE. Specifically, our work reveals that SPLs can be relevant to the particularities of GSE: generating new video game content (one of the hot topics of video game research), making software accessible to multidisciplinary teams, or balancing difficulty (one of the seminal problems of video games).

The remainder of the paper is structured as follows. Section 1.2 presents the video game Kromaia and the two development paradigms studied (MDD and CDD). Section 1.3 describes our experiment. Section 1.4 presents the results obtained. Section 1.5 discusses the findings. Section 1.6 describes the threats to validity. Section 1.7 examines the related work of the area. Finally, Section 1.8 concludes the paper.

1.2 Kromaia and its paradigms

This section presents Kromaia¹ and the multiple processes available to generate bosses applying the two development approaches (CaO or SPL) under the two paradigms studied (MDD and CDD).

Kromaia is a three-dimensional space game created by Kraken Empire, in which the player's spaceship flies from a starting point to an ending point, reaching the goal of each level before being destroyed. Throughout each level, the player has to explore floating structures, avoid asteroids and find items. Along the way, the player will encounter basic enemies trying to damage the

¹Kromaia Omega - Launch | PlayStation 4: <https://youtu.be/EhsejJBp8Go>

player's spaceship by firing projectiles. If the player reaches the destination, the final boss corresponding to that level appears and must be defeated in order to complete the level.

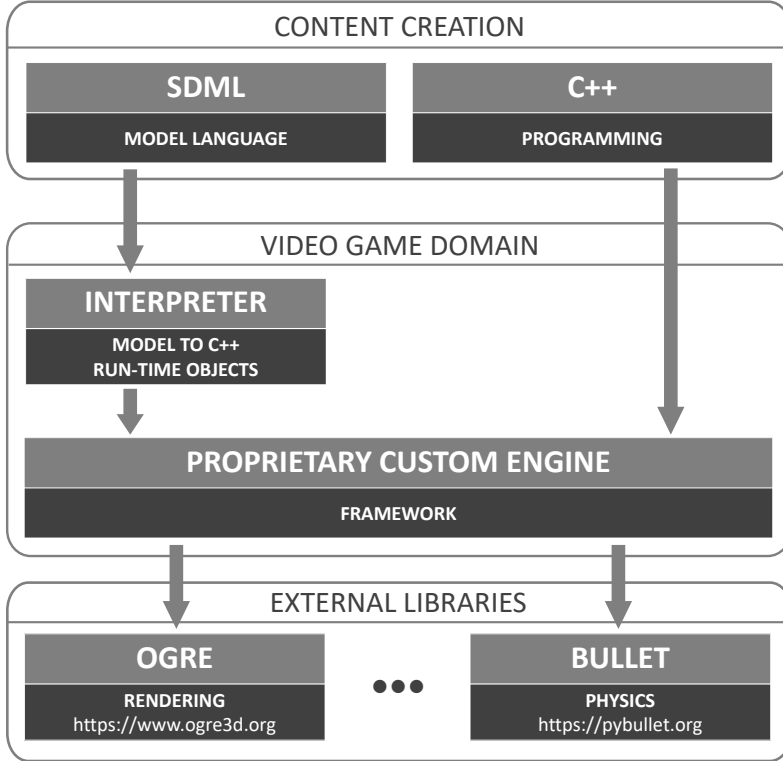


Figure 1.1: Architecture of Kromaia. Adapted from [6]

Figure 1.1 presents an overview of the architecture of Kromaia. The top layer (content creation) allows the developers to add new content to the video game in two different ways, using models (SDML) or source code (C++):

- **SDML:** The Shooter Definition Modeling Language, created by the developers of the game, allows the creation of any type of element that is present in the game. It will be used in the experiment under the MDD paradigm.
- **C++:** The C++ programming language can also be used to create content for the game. Game elements created in C++ must follow some

structure rules and patterns (e.g., inherit from a specific class) in order to be properly integrated with the rest of the game.

The middle layer (video game domain) contains the core of the game and its main functionality. When the content is provided as an SDML model, it will be interpreted at run-time and transformed into a C++ run-time object. When the content is provided in the form of source code, it will interact directly with the proprietary custom engine. The engine contains the base classes and functionality used by the game elements. The developers of the game also created an Application Programming Interface (API) to facilitate the creation of new content and its integration with the rest of the game.

The bottom layer (external libraries) contains the set of libraries used for specific purposes, such as the rendering of the game (Ogre) or the management of the physics (Bullet). The engine is built upon those libraries and will interact with them when needed.

Below, we present how the two development paradigms studied (MDD and CDD) are realized in the Kromaia architecture.

1.2.1 The MDD paradigm in Kromaia

Figure 1.2 shows the main artifacts used under the MDD paradigm for developing final bosses in Kromaia. Bosses are specified using the Shooter Definition Modeling Language (SDML) [6]. SDML is a Domain-Specific Language (DSL) that defines components that appear in video game entities: the anatomical structure (including which parts are used in it, their physical properties, and how they are connected); the amount and distribution of vulnerable parts, weapons, and defenses in the structure/body of the character; and the movement behaviours associated to the whole body or its parts. This modeling language has concepts such as hulls, links, weak points, and weapons.

The top part of Figure 1.2 shows a simplified version of the SDML metamodel and its concrete syntax. The complete metamodel contains more than 20 concepts, over 20 relationships, and more than 60 properties [6]. However, this simplified version is complete enough to understand the elements that compose the structure of a boss and their relationships with each other. The figure depicts the concrete syntax of each of the elements. Overall, a boss is composed of hulls and links. Each link joins two hulls. A hull can contain weak points that the player must attack, weapons such as cannons, lasers, or spikes, and shields that protect weak points and weapons.

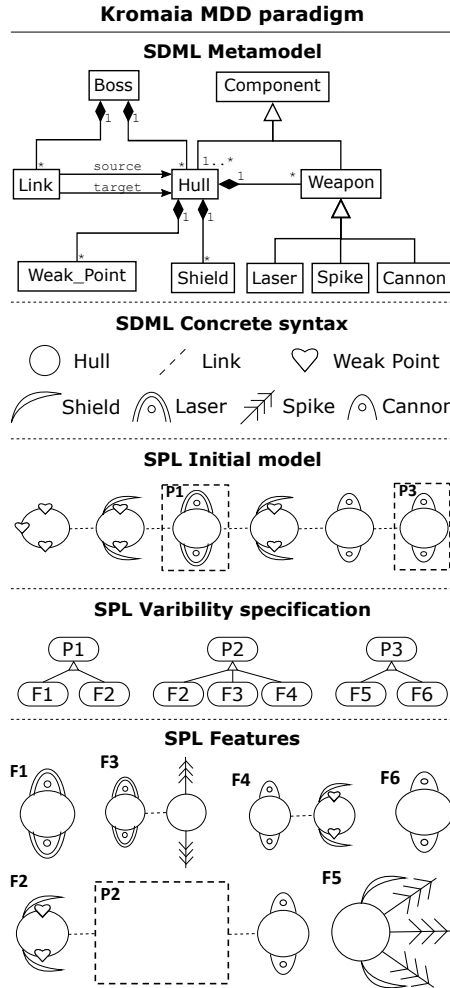


Figure 1.2: Kromaia MDD paradigm: the Shooter Definition Modeling Language (SDML) (Simplified version) and a subset of the SPL used to develop models of bosses under the MDD paradigm.

An actual example of a final boss of Kromaia is presented in the top part of Figure 1.3. The Serpent is the final boss that the player must defeat in order to complete Level 1. The middle part of the figure shows the model of the boss using the SDML concrete syntax. This boss is composed of a series of nine hulls that are linked in a line. The first hull is the head, which includes three weak points and will drive the movement of the boss. The next seven hulls

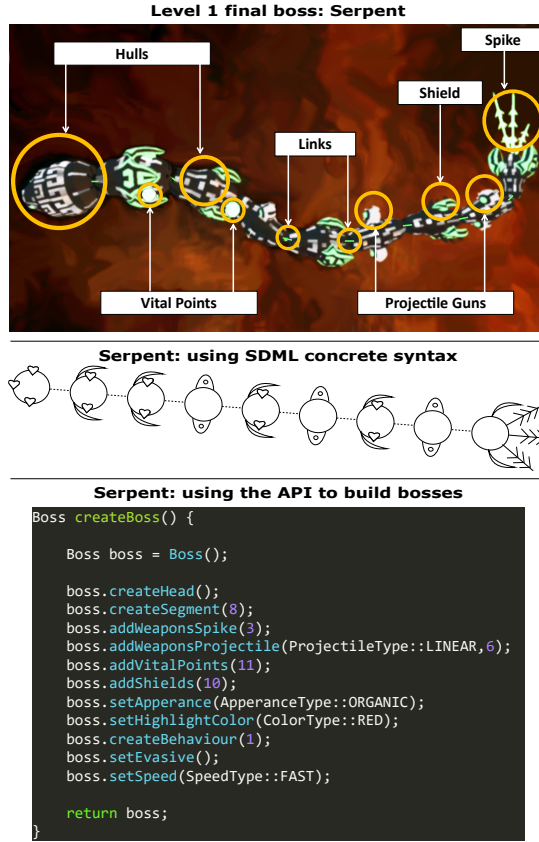


Figure 1.3: Example of a final boss in Kromaia.

contain either two vital points covered with shields or two cannons. The last hull is the tail, containing two shields and three spikes that are used to hit the player.

Under the MDD paradigm, when bosses are created using the CaO development approach, the developer has access to a set of SDML models that have been created in the past (such as the serpent in the middle part of Figure 1.3). The developer can use them to serve as the basis for the new boss being created or can reuse parts of the existing models to build the new boss. For example, the developer could copy the existing serpent model and add some new weapons to one of the existing hulls or copy the last two hulls and connect them to create a two-tailed serpent boss.

Under the MDD paradigm, when bosses are created using the SPL development approach, the developer has access to a feature-oriented SPL that can be used to generate SDML models. The SPL follows a compositional approach [25]. The developer will begin with an initial model and perform some substitutions of model fragments using a library of model fragments and following the rules specified by the variability specification.

The middle part of Figure 1.2 shows a subset of the assets of the SPL for developing Kromaia bosses. The figure shows one of the initial models that can be used to create new bosses. The bottom of Figure 1.2 shows some feature models that represent the variability specification and some of the features present in the library of the SPL. Each feature corresponds to a model fragment that is expressed using the concrete syntax of SDML.

The domain experts of Kromaia defined the variability specification. The elements denoted with a P represent variation points that can be substituted by a feature, F , following the variability specification. For example, the variability specification that is shown in the first position ($P1$) indicates that it can be substituted with the model fragments $F1$ or $F2$. Furthermore, some features can include variation points that must be fulfilled with another feature. For example, feature $F2$ has a variation point, $P2$, which has to be substituted following its variability specification (by $F2$, $F3$ or $F4$).

1.2.2 The CDD paradigm

Figure 1.4 shows the main artifacts used when creating a boss under the CDD paradigm. The bottom part of Figure 1.3 shows an example of a boss created under the CDD paradigm, where the final bosses are specified using the C++ programming language and the API of Kromaia [6].

The API contains a set of base classes that can be extended when creating new content. It also contains functionality shared by many elements of the video game. The top part of Figure 1.4 shows a subset of the API for creating bosses in Kromaia. It is structured in different sections that include: functions to create a series of hulls linked together following different forms (rings, segments, squares) and different sizes; functions to add weapons to existing hulls and to manage the properties of those weapons (e.g., type of projectile used); functions to include vital points and shields to the different hulls; and functions to determine the appearance (e.g., predominant color for the hulls) and behaviour (e.g., different IA parameters that determine the actions taken in the game) of the boss being created.

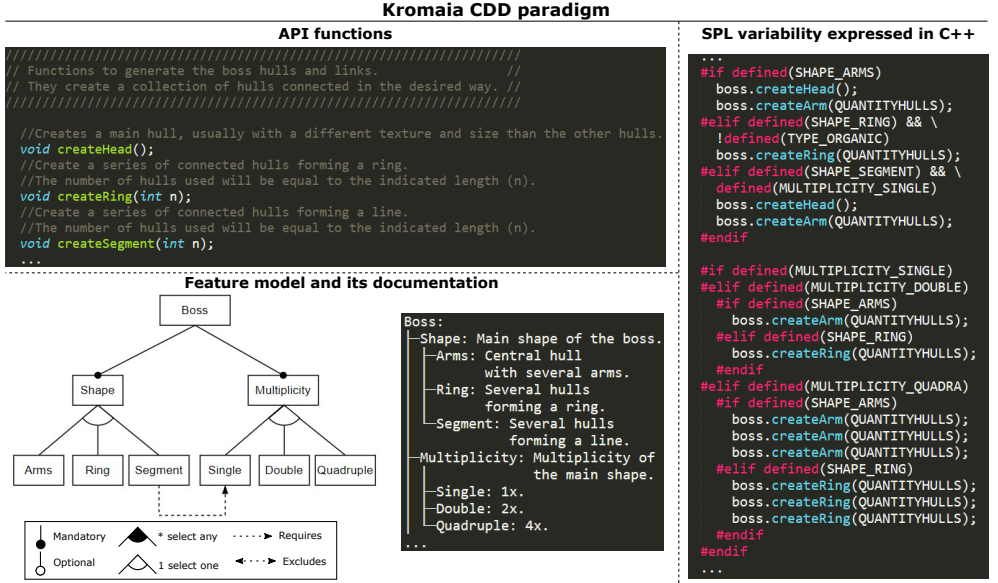


Figure 1.4: Kromaia CDD paradigm: a subset of the API functions and a subset of the SPL used to develop bosses under the CDD paradigm.

Under the CDD paradigm, when bosses are created using the CaO development approach, the developer has access to the API, its documentation, and a set of source codes of other bosses that have been created in the past (such as the Serpent shown at the bottom part of Figure 1.3). The developer will use them in the same way as in the MDD paradigm by serving as the basis for the new boss being created, reusing parts of the existing code to build the new boss, adding new lines of code, or calling the functions in the API.

Under the CDD paradigm, when bosses are created using the SPL development approach, the developer has access to an annotative SPL [25] that can be used to generate the source code of the bosses (see the bottom and right part of Figure 1.4). The developer will derive new products through the feature model, creating a configuration that will then be used to generate the source code. The developer has access to the feature model itself, the documentation of each of the features present in it, and the annotated source code that is being configured through the feature model.

Figure 1.4 shows part of the feature model driving the SPL under the CDD paradigm. It determines the different features that can be selected when creating a boss from the SPL and some constraints about the choices that can be

made (e.g., having a segment shape implies a single multiplicity). The documentation includes a description of each of the features that can or cannot be included in the boss being created. When a configuration of a product is done, it will be used to generate the source code of the boss following an annotative approach. Each of the features will produce a set of *define* macros that will be used to pre-compile a C++ source code that includes the information about all the bosses (see Figure 1.4, right). Depending on the features selected and the *#define* macros generated, a different set of C++ lines will be included in the resulting C++ code. For example, if the feature *Arms* and the feature *Quadruple* are selected, the macro *define SHAPE_ARMS* and *define MULTIPLICITY_QUADRA* will be used to pre-compile the source code, resulting in the line *boss.createArm(x)* being included four times in the final source code (the *QUANTITYHULLS* is defined by another feature that is related to the size of the boss being created).

1.3 Experiment design

This section presents the experimental design of both experiments. It includes information about the objectives, the variables studied, the research questions, the design chosen, the participants recruited, the experimental objects used during the experiments, the procedure followed in each execution of the experiments, and the statistical analysis selected for the treatment of the data collected.

1.3.1 Objectives

According to Wohlin's guidelines [54] for reporting software engineering experiments, we have organized our research objectives using the Goal Question Metric template for goal definition [4].

Our goal is to **analyze** different development approaches **for the purpose of** comparison, **with respect to** the correctness, efficiency, and user satisfaction of the software constructed, **from the point of view of** the experience of the developer (inexperienced or experienced), **in the context of** creating software for video games under two different development paradigms: MDD and CDD.

1.3.2 Variables

In this study, the factor under investigation is the Development Approach (DA). We investigate two alternatives of the factor: create the final bosses of a video game using CaO or using an SPL.

To evaluate the effects of the use of the different approaches, we selected Correctness and Efficiency (related to the subjects' performance) as objective dependent variables, and user Satisfaction (related to the subjects' perception) as subjective dependent variable.

We measured Correctness using a correction template, which was applied to the bosses developed by the subjects after the experiment. Their values range from 0 to 100 and represent the percentage of points obtained according to the correction template.

To calculate Efficiency, we measured the time employed by each subject to finish the task, using the start and end times of each task. Efficiency is the ratio of Correctness to time spent (in minutes) to perform a task.

We measured Satisfaction using a 5-point Likert-scale questionnaire based on the Technology Acceptance Model (TAM) [37]. We decompose Satisfaction into three subjective dependent variables as follows:

- Perceived Ease of Use (PEOU), the degree to which a person believes that learning and using a particular DA would require less effort.
- Perceived Usefulness (PU), the degree to which a person believes that using a particular DA will increase performance.
- Intention To Use (ITU), the degree to which a person intends to use a DA.

Each of these variables corresponds to specific items in the TAM questionnaire. We average the scores obtained for these items to obtain the value for each variable.

To take into account the effects that the different development paradigms could have on the DA, we considered the Paradigm blocking factor with two alternatives: CDD and MDD. We carried out two experiments with the same design, each one based on a different development paradigm. The first experiment involves GSE tasks under the MDD paradigm, while the second experiment involves GSE tasks under the CDD paradigm.

1.3.3 Research questions and hypotheses

The research questions and null hypotheses are formulated as follows:

RQ1 - Does the DA used to create software for video games impact the Correctness of the software? The corresponding null hypothesis is $H_{0,C}$: The DA used for creating software for video games does not have an effect on Correctness.

RQ2 - Does the DA used to create software for video games impact the Efficiency of developers? The null hypothesis for Efficiency is $H_{0,E}$: The DA does not have an effect on Efficiency.

RQ3 - Is user satisfaction different when developers use different DA to create software for video games? To answer this question, we formulated three hypotheses based on the variables *PEOU*, *PU*, and *ITU* with their corresponding null hypotheses:

$H_{0,PEOU}$ - The DA does not have an effect on *PEOU*.

$H_{0,PU}$ - The DA does not have an effect on *PU*.

$H_{0,ITU}$ - The DA does not have an effect on *ITU*.

RQ4 - Are the answers to RQ1, RQ2, and RQ3 the same when using different development paradigms (MDD and CDD)? To address this question, we will answer RQ1, RQ2, and RQ3 for each of the two paradigms considered.

1.3.4 Design

For the two experiments of this study, we use a factorial crossover design with two periods, using different tasks (T1 and T2) for each period. The subjects are randomly divided into two groups (G1 and G2). In the first period of the experiment, all of the subjects perform T1, with subjects in G1 using CaO and subjects in G2 using SPL. Then, in the second period of the experiment, all of the subjects perform T2, with subjects in G1 using the SPL and subjects in G2 using CaO.

The repeated measures design increases the sensitivity of the experiment [52]: the observation of the same subject using the two alternatives controls between-subject differences, improving experiment robustness regarding variation among subjects. By using two different sequences for each group (G1 uses CaO first and SPL second; G2 uses SPL first and CaO second) and different tasks, the

design counterbalances some of the effects that can be caused by the order in which development approaches are used (i.e., learning effect, fatigue).

Before conducting each experiment, to verify the experiment design, we conducted a pilot study with two subjects. The pilot study allowed us to estimate the time needed to perform the tasks and complete the questionnaires, detect typographical and semantic errors, and test the instruments used to collect the data and conduct the experiment. The two subjects were different for each pilot study, and they did not participate in the experiments.

1.3.5 Participants

We selected the subjects using convenience sampling [54] via an open call to volunteers. A total of 79 subjects with different knowledge about programming, modeling, and video game development performed the experiment; 28 of them performed the experiment under the MDD paradigm and 51 performed the experiment under the CDD paradigm. Table 1.1 presents detailed demographic data about the subjects.

Each execution of the experiment was conducted by two instructors and one expert in video game development. The expert provided information about the Kromaia framework and examples of how to create bosses using CaO and the SPL under the paradigm studied in the experiment. During the experiment, one of the instructors gave instructions and managed the focus groups, while the other clarified doubts for the subjects and took notes.

1.3.6 Experimental objects

The tasks of our experiments consisted in developing two final bosses of Kromaia from a gameplay video showing its structure and behavior. A video game software engineer involved in the development of Kromaia and a researcher designed the tasks, balanced the complexity (to ensure similar difficulty across tasks), and prepared the correction templates.

The subjects got access to training material with examples of how to create a boss under the paradigm being studied (MDD or CDD) using CaO or the SPL. Depending on the DA and paradigm being used, the subjects got access to different artifacts to create the bosses (see Section 1.2).

For the data collection, we prepared a set of two forms for each experiment (one for each experimental sequence) with the following sections: (I) an in-

formed consent statement that the subjects had to review and accept voluntarily, which clearly explained what the experiment consisted of and what will be the treatment given to personal data; (II) a demographic questionnaire to characterize the sample; and (III) a specific questionnaire to collect the subjects' responses during the experiment (their tasks, their times, and their answers to the satisfaction questionnaire).

The experimental objects used in these experiments (the training material, the tasks, and the forms used for the questionnaires), as well as the results and the statistical analysis, are available to the reader².

1.3.7 Experimental procedure

Each of the experiments performed for this study was conducted on two different days. On one of the days, the experiment was conducted online with professionals (experienced subjects). On the other day, the experiment was conducted face-to-face with the group of students (inexperienced subjects). All of the sessions were scheduled for a duration of one hour and 45 minutes and were carried out following the experimental procedure described below:

1. (~ 5 min) An instructor explained the parts of the session and clarified that it was not a test of their abilities.
2. (~10 min) The subjects attended a tutorial about the video game bosses to be created and how to use CaO or SPL to create these bosses under the paradigm studied in the experiment (MDD or CDD).
3. (~5 min) The subjects received clear instructions on how to access the form for the experiment and the artifacts needed to complete the task. The subjects were randomly divided into two groups (G1 and G2) and each group got access to the corresponding form.
4. (~5 min) The subjects accessed the form and then read and confirmed having read the information about the experiment, the data treatment of their personal information, and the voluntary nature of their participation before accessing the questionnaires and tasks of the experiment.
5. (~5 min) The subjects completed a demographic questionnaire.
6. (~30 min) The subjects performed the first task. The subjects from G1 had to use the CaO approach to create a given boss of the video game,

² Available at <http://svit.usj.es/SPLvsCAO-MDD-CDD>

and the subjects from G2 had to create the same boss but using the SPL. After submitting their solutions, the subjects completed a satisfaction questionnaire about the approach used for development.

7. (~30 min) The subjects performed the second task, which has the creation of another boss. This time, the subjects from G1 created it using the SPL approach, while the subjects from G2 created it using the CaO approach. Then, the subjects completed the satisfaction questionnaire.
8. (~15 min) A focus group interview about the tasks was conducted by the instructors.

1.3.8 Analysis procedure

To analyze the data extracted from the experiments, we performed a statistical study of the aggregated data from the two experiments, and we also performed a statistical study of the data extracted from each experiment separately. The analysis procedure was the same for all three data sets. We chose the Linear Mixed Model (LMM) [53] for the statistical data analysis. LMM handles correlated data resulting from repeated measurements, and it allows us to study the effects of factors that intervene in a crossover design (period and sequence) and the effects of other blocking factors (Paradigm or Experience) [52].

In the hypothesis testing, we applied the Type III test of fixed effects with unstructured repeated covariance. Type III is the default test, which enables LMM to produce the exact F-values and p-values for each dependent variable and each fixed factor. The assumption for applying LMM is the normality of the residuals of the dependent variables. To verify this normality, we used Kolmogorov-Smirnov tests as well as visual inspections of the histogram and normal Q-Q plots.

In this study, DA was defined as a fixed-repeated factor to identify the differences between using CaO or SPL, and the subjects were defined as a random factor ($1|Subj.$) to reflect the variability among subjects. The dependent variables (DV) for this test were Correctness and Efficiency, and the three subjective variables related to Satisfaction: PEOU, PU, and ITU.

To analyze the potential effects of the development paradigm to determine the variability due to DA in the dependent variables, in the statistical models used for analyzing the aggregated data, we considered the Paradigm fixed factor and the combination of this factor with DA. In the statistical models used to analyze the three data sets, we also considered Period and Sequence as fixed

factors to take into account the potential effects of factors that intervene in a crossover design. In order to take into account the potential effects of the subjects' experience to determine the variability in the dependent variables, we also considered Experience as a fixed factor and its combinations with the other fixed factors considered in the statistical models (DA, Paradigm, Sequence, and Period).

We tested different statistical models to find out which factors, in addition to DA, could best explain the changes in the dependent variables. Some of these statistical models are described mathematically in Formula 1.1. The starting statistical model (Model 0) reflects the main factor used in this experiment (DA) and the Subject random factor ($1|Subj.$). We also tested other statistical models (e.g., models M_1 , M_2 , M_3 , or M_4) that included other fixed factors (Paradigm, Experience, Period, or Sequence), or their combinations, which could have effects on the dependent variables.

$$\begin{aligned}
 (M_0) \quad DV &\sim DA + (1|Subj.) \\
 (M_1) \quad DV &\sim DA + Paradigm + DA * Paradigm + (1|Subj.) \\
 (M_2) \quad DV &\sim DA + Paradigm + Experience + (1|Subj.) \\
 (M_3) \quad DV &\sim DA + Experience + DA * Experience + (1|Subj.) \\
 (M_4) \quad DV &\sim DA + Sequence + Period + (1|Subj.)
 \end{aligned} \tag{1.1}$$

The statistical model fit of the tested models was evaluated based on goodness of fit measures such as Akaike's information criterion (AIC) and Schwarz's Bayesian Information Criterion (BIC). The model with the smallest AIC or BIC is considered to be the best fitting model [24, 16]. In the analysis, we only considered statistical models verifying the normality of the residuals of the dependent variables. Therefore, to describe the changes in each dependent variable, we selected the statistical model that obtained the smallest AIC or BIC value from the statistical models that satisfied the normality of residuals.

To quantify the differences in the dependent variables due to significant fixed factors, we calculated the Cohen d value [10] between the alternatives of these factors. Cohen d values between 0.2 and 0.3 indicate a small effect, values around 0.5 indicate a medium effect, and values greater than 0.8 indicate a large effect. We selected histograms and box plots to describe the data and the results graphically.

1.4 Results

The subjects filled out a demographic questionnaire that was used for characterizing the sample. Table 1.1 shows the number of subjects of each experiment grouped by experience, “inexperienced” subjects were students, and “experienced” subjects were professionals in the game development industry. The table shows the mean and standard deviation of age, hours per day developing software (Developing time), and hours per day working with models (Modeling time). A 5-point Likert-scale was used for the self-assessment of the subjects’ knowledge of programming languages (Programming knowledge) and modeling languages (Modeling Knowledge), which are also shown in Table 1.1.

Table 1.1: Results of the demographic questionnaire

	Number of subjects	Age $\mu \pm \sigma$	Developing time $\pm\sigma$	Modeling time $\pm\sigma$	Programming knowledge $\pm\sigma$	Modeling knowledge $\pm\sigma$
All subjects	79	23.5 $\pm$ 6.3	2.5 $\pm$ 2.2	0.7 $\pm$ 1	3.4 $\pm$ 1.2	2.8 $\pm$ 1.2
MDD subjects	28	25.8 $\pm$ 7.4	3.4 $\pm$ 2.6	0.8 $\pm$ 1	4 $\pm$ 1.1	3 $\pm$ 1.3
CDD subjects	51	22.3 $\pm$ 5.3	2 $\pm$ 1.8	0.6 $\pm$ 0.9	3.1 $\pm$ 1.1	2.7 $\pm$ 1.1
Experienced	26	29.9 $\pm$ 7.2	3.8 $\pm$ 2.9	1.2 $\pm$ 1.2	4.3 $\pm$ 0.9	3.5 $\pm$ 1.3
MDD Experienced	13	30.7 $\pm$ 8.4	4.5 $\pm$ 2.7	1.2 $\pm$ 1.2	4.6 $\pm$ 0.8	3.8 $\pm$ 1.4
CDD Experienced	13	29.2 $\pm$ 6.1	3.2 $\pm$ 2.9	1.2 $\pm$ 1.3	4.1 $\pm$ 1	3.3 $\pm$ 1.3
Inexperienced	53	20.3 $\pm$ 1.6	1.9 $\pm$ 1.4	0.4 $\pm$ 0.6	2.9 $\pm$ 1	2.4 $\pm$ 0.9
MDD Inexperienced	15	21.5 $\pm$ 1.3	2.4 $\pm$ 2	0.5 $\pm$ 0.6	3.5 $\pm$ 1.1	2.3 $\pm$ 0.7
CCD Inexperienced	38	19.8 $\pm$ 1.5	1.7 $\pm$ 1	0.4 $\pm$ 0.6	2.7 $\pm$ 0.9	2.5 $\pm$ 1

1.4.1 Changes in the dependent variables

The results of the two experiments are shown in Table 1.2. The first set of rows (ALL) corresponds to the aggregated data of the two experiments; the second and third sets of rows correspond to the data of each experiment individually (CDD and MDD). The table also shows the values for the mean and standard deviation of the dependent variables Correctness, Efficiency, PEOU, PU, and ITU for each of the DAs compared (CaO and SPL) and for each of the alternatives for the fixed factors considered. These include Experience, with two alternatives (Experienced and Inexperienced subjects); Period, with two alternatives (Task 1, and Task 2); and Sequence, whose two alternatives reflect the order in which subjects used the DAs, (G1: CaO-SPL, G2: SPL-CaO). The aggregated data (ALL) should also contain a column with the Paradigm fixed factor and its two alternatives CDD and MDD, but it has been removed to

avoid duplicity (those values are also shown under the corresponding rows of CDD and MDD)

Taking into account the aggregated data from the two experiments performed in this study, we can state that there were differences in the means and standard deviations of all of the dependent variables depending on which DA was used to create the boss. If we compare the statistical results obtained in each paradigm separately, we can state that the differences in the dependent variables, depending on which DA was used to create a boss of a video game, were smaller under the MDD paradigm than under the CDD paradigm.

Table 1.2: Values for the mean and standard deviation ($\mu \pm \sigma$) of the dependent variables for the factor Development Approach in each alternative of the fixed factors for data aggregated (ALL), data under the CDD paradigm and data under the MDD paradigm. Correctness (%) [0-100]; Efficiency (%/min)[0-100]; PEOU,PU,ITU [0-5]

		Development Approach		Experience		Period		Sequence	
				Experienced	Inexperienced	Task 1	Task 2	G1 CaO-SPL	G2 SPL-CaO
ALL	Correctness	CaO	53.15±16.98	62.42±10.55	48.52±17.74	53.98±17.87	52.32±16.22	53.98±17.87	52.32±16.22
		SPL	67.75±20.05	75.69±17.93	64.01±20.06	67.91±15.48	67.59±23.98	67.59±23.98	67.91±15.48
	Efficiency	CaO	2.88±1.49	3.37±1.22	2.64±1.57	2.45±1.12	3.31±1.7	2.45±1.12	3.31±1.7
		SPL	3.68±1.69	4.02±1.45	3.53±1.79	3.06±0.97	4.31±2.01	4.31±2.01	3.06±0.97
	PEOU	CaO	3.56±0.85	3.78±0.84	3.45±0.84	3.59±0.88	3.53±0.84	3.59±0.88	3.53±0.84
		SPL	3.97±0.84	4.19±0.95	3.86±0.77	4.18±0.69	3.77±0.94	3.77±0.94	4.18±0.69
	PU	CaO	3.65±0.78	3.88±0.77	3.53±0.77	3.69±0.83	3.6±0.74	3.69±0.83	3.6±0.74
		SPL	3.89±0.76	4.11±0.8	3.78±0.72	4±0.7	3.78±0.8	3.78±0.8	4±0.7
	ITU	CaO	3.25±1.12	3.46±1.12	3.14±1.11	3.3±1.25	3.19±0.97	3.3±1.25	3.19±0.97
		SPL	3.51±1.13	3.69±1.13	3.43±1.13	3.85±0.89	3.19±1.25	3.19±1.25	3.85±0.89
CDD	Correctness	CaO	53.87±17.28	64.69±8.14	50.06±18.09	55.93±17.64	51.97±17.07	55.93±17.64	51.97±17.07
		SPL	69.78±11.21	74.83±11.14	68.18±10.9	67.03±11.49	72.75±10.33	72.75±10.33	67.03±11.49
	Efficiency	CaO	2.58±0.92	3.07±0.67	2.41±0.95	2.43±0.97	2.72±0.87	2.43±0.97	2.72±0.87
		SPL	3.9±1.61	4.13±0.94	3.83±1.77	2.95±0.73	4.94±1.66	4.94±1.66	2.95±0.73
	PEOU	CaO	3.57±0.84	3.72±0.71	3.51±0.88	3.51±0.98	3.62±0.7	3.51±0.98	3.62±0.7
		SPL	4.07±0.7	4.45±0.54	3.94±0.7	4.13±0.72	4.01±0.68	4.01±0.68	4.13±0.72
	PU	CaO	3.56±0.8	3.58±0.73	3.56±0.83	3.57±0.89	3.55±0.72	3.57±0.89	3.55±0.72
		SPL	3.99±0.65	4.31±0.48	3.88±0.66	4.02±0.68	3.97±0.63	3.97±0.63	4.02±0.68
	ITU	CaO	3.22±1.11	3±1.17	3.29±1.1	3.18±1.25	3.25±0.99	3.18±1.25	3.25±0.99
		SPL	3.65±1.04	3.73±1.07	3.62±1.04	3.85±0.9	3.44±1.15	3.44±1.15	3.85±0.9
MDD	Correctness	CaO	51.87±16.64	60.15±12.43	44.7±16.82	50.87±18.39	53.03±15.02	50.87±18.39	53.03±15.02
		SPL	64.13±29.96	76.48±22.98	53.43±31.84	69.66±21.88	59.34±35.6	59.34±35.6	69.66±21.88
	Efficiency	CaO	3.41±2.09	3.67±1.57	3.19±2.49	2.47±1.37	4.5±2.29	2.47±1.37	4.5±2.29
		SPL	3.29±1.8	3.91±1.83	2.76±1.64	3.27±1.34	3.31±2.16	3.31±2.16	3.27±1.34
	PEOU	CaO	3.54±0.88	3.85±0.97	3.28±0.73	3.72±0.67	3.33±1.07	3.72±0.67	3.33±1.07
		SPL	3.78±1.05	3.92±1.2	3.67±0.93	4.27±0.63	3.37±1.18	3.37±1.18	4.27±0.63
	PU	CaO	3.8±0.74	4.18±0.71	3.5±0.7	3.89±0.69	3.7±0.82	3.89±0.69	3.7±0.82
		SPL	3.7±0.9	3.91±1.01	3.48±0.62	3.97±0.78	3.47±0.96	3.47±0.96	3.97±0.78
	ITU	CaO	3.3±1.14	3.92±0.89	2.77±1.08	.5±1.28	3.08±0.95	3.5±1.28	3.08±0.95
		SPL	3.27±1.26	3.65±1.23	2.93±1.22	3.85±0.9	2.77±1.33	2.77±1.33	3.85±0.9

To quantify the differences in the dependent variables, we analyzed the Cohen d values for each of the factors considered in the study. Table 1.3 shows the Cohen d value of each factor for each dependent variable of the aggregate data

of the two experiments performed. Table 1.4 and Table 1.5 show the same information but particularized to the data extracted in each of the experiments under the CDD paradigm and the MDD paradigm, respectively. Values indicating high, medium, and small variations due to the factor are shaded in dark grey, grey, and light grey, respectively. Those values corresponding to significant differences according to the hypothesis tests are highlighted in cursive. Positive values indicate differences in favor of the first alternative of the factors, while negative values indicate differences in favor of the second alternative.

According to the Cohen d values of the dependent variables for the DA of the aggregated data (the first column of Table 1.3), the data under the CDD paradigm (the first column of Table 1.4), and the data under the MDD paradigm (the first column of Table 1.5), we can affirm that the differences due to the DA factor were large in Correctness, medium in Efficiency and PEOU, and small in PU and in ITU. In addition, we can state that the differences due to the DA factor were larger under the CDD paradigm than under the MDD paradigm for all of the dependent variables considered in the experiment. Under the CDD paradigm, the effect of DA on Correctness and Efficiency was large and in favor of the SPL approach. In contrast, under the MDD paradigm, the effect of DA on Correctness was medium in favor of SPL and the effect on Efficiency was negligible in favor of CaO. The effect of DA on Satisfaction was also different depending on the paradigm considered: under the CDD paradigm, the effect of DA on Satisfaction was medium-large in favor of SPL; in contrast, under the MDD paradigm, the effect of DA was small for PEOU in favor of SPL, and very small or negligible for PU and ITU in favor of CaO.

The third column of Table 1.3, and the second columns of Tables 1.4 and 1.5 show how the Experience factor had large effects on Correctness, medium effects on Efficiency, and PEOU, and different effects on PU and ITU depending on the paradigm considered. The experienced subjects obtained better results in terms of correctness and efficiency than the inexperienced subjects and were more generous in rating their satisfaction with the approaches used in the experiment. The differences were greater between the experienced and inexperienced subjects under the MDD paradigm than under the CDD paradigm. Under both paradigms, the effect size of Experience for the dependent variables was in the same range for all of the dependent variables except for PU and ITU. Under the CDD paradigm, the experienced subjects rated slightly higher PU than the inexperienced subjects; however, both groups of subjects reported similar values for ITU. In contrast, under the MDD paradigm, the

differences between the ratings of experienced and inexperienced subjects were large in favor of the experienced subjects for both PU and ITU.

The Period and Sequence factors also had a different effect on the dependent variables depending on the paradigm under which the subjects performed the experiment (see the last two columns of Tables 1.3, 1.4, and 1.5). Under the CDD paradigm, the Period factor had large effects on Efficiency, but the effects on the other dependent variables were small or negligible. However, under the MDD paradigm, the Period factor had medium-large effects on all of the dependent variables except Correctness. In both experiments, the subjects built bosses of similar correctness in both tasks, but the subjects were more efficient in the second task and rated their satisfaction higher with the approach used on the first task. Under both paradigms, the effect size of Sequence was small for Correctness and medium-large for Efficiency; however, the effects were the opposite depending on the paradigm. Under the CDD paradigm, the subjects in G1 (who started performing the tasks with CaO) obtained better results in Correctness and Efficiency, while under the MDD paradigm, the subjects in G2 (who started performing the tasks with SPL) obtained better results in Correctness and Efficiency. Under both paradigms, the subjects who started with SPL reported being more satisfied than the subjects who started with CaO.

Table 1.3: Cohen d values for the independent variables for each fixed factor.

ALL	Development Approach (CaO/SPL)	Paradigm (CDD/MDD)	Experience (Experienced/ Inexperienced)	Period (Task 1/ Task 2)	Sequence (G1(CaO-SPL)/ G2(SPL-CaO))
<i>Correctness</i>	-0,786	0,181	0,687	0,049	0,034
<i>Efficiency</i>	-0,503	-0,065	0,387	-0,682	0,119
<i>PEOU</i>	-0,485	0,172	0,378	0,266	-0,2
<i>PU</i>	-0,317	0,034	0,432	0,2	-0,086
<i>ITU</i>	-0,237	0,127	0,262	0,341	-0,245

1.4.2 Answers to the Research Questions

Table 1.6 shows the results of the Type III fixed effects test applied on the selected statistical models for each of the dependent variables and for each fixed factor considering the aggregated data. Tables 1.7 and 1.8 show the same information as Table 1.6 but related only to the data extracted during each of the experiments, under the CDD and the MDD paradigm, respectively. The values indicating significant differences are shaded in grey. The statistical model selected for each variable consists of the factors for which statistical

Table 1.4: Cohen d values for the independent variables for each fixed factor under the CDD paradigm.

CDD	Development Approach (CaO/SPL)	Experience (Experienced/ Inexperienced)	Period (Task 1/ Task 2)	Sequence (G1(CaO-SPL)/ /G2(SPL-CaO))
<i>Correctness</i>	-1.092	0.713	-0.014	0.294
<i>Efficiency</i>	-1.008	0.344	-0.794	0.599
<i>PEOU</i>	-0.651	0.462	0.016	-0.146
<i>PU</i>	-0.594	0.301	0.058	-0.024
<i>ITU</i>	-0.4	-0.079	0.161	-0.217

Table 1.5: Cohen d values for the independent variables for each fixed factor under the MDD paradigm

MDD	Development Approach (CaO/SPL)	Experience (Experienced/ Inexperienced)	Period (Task 1/ Task 2)	Sequence (G1(CaO-SPL)/ /G2(SPL-CaO))
<i>Correctness</i>	-0.506	0.843	0.127	-0.254
<i>Efficiency</i>	0.062	0.433	-0.544	-0.529
<i>PEOU</i>	-0.251	0.428	0.675	-0.265
<i>PU</i>	0.119	0.684	0.441	-0.186
<i>ITU</i>	0.030	0.853	0.658	-0.279

Table 1.6: Results of the Type III test of fixed effects for each dependent variable and factor. NA=Not Applicable

ALL	Development Approach (DA)	Paradigm	Experience (Exp)	Period	Sequence
<i>Correctness</i>	F=33.284; p<.001	F=2.963; p=0.089	F=19.644; p<.001	F=0.31; p=0.579	F=0.09; p=0.765
<i>Efficiency</i>	F=7.664; p=0.007	F=0.017; p=0.895	F=4.163; p=0.045	NA	F=0.83; p=0.366
<i>PEOU</i>	F=16.765; p<.001	F=2.367; p=0.128	F=5.54; p=0.021	F=9.843; p=0.002	NA
<i>PU</i>	F=2.127; p=0.149	F=0.21; p=0.648	F=7.293; p=0.009	NA	NA
<i>ITU</i>	F=1.607; p=0.209	F=0.193; p=0.662	F=5.217; p=0.025	F=6.833; p=0.011	F=2.439; p=0.123
	DA*Paradigm	Paradigm*Exp	Paradigm*Period	Paradigm*Sequence	
<i>Correctness</i>	F=0.577; p=0.45	F=1.03; p=0.314	F=1.146; p=0.288	F=2.704; p=0.104	
<i>Efficiency</i>	F=11.167; p=0.001	F=0.279; p=0.599	NA	F=10.825; p=0.002	
<i>PEOU</i>	NA	NA	F=9.118; p=0.003	NA	
<i>PU</i>	F=5.376; p=0.023	F=1.128; p=0.292	NA	NA	
<i>ITU</i>	F=1.358; p=0.248	F=6.956; p=0.01	F=2.764; p=0.101	NA	

significance is shown in Tables 1.6, 1.7, and 1.8. If NA appears, it is because the factor in that column is not part of the fixed factors considered in the statistical model selected for the variable in that row. For example, the model selected for Correctness contains all of the fixed factors that appear as headings in the table and the random factor. Its mathematical representation would be

Table 1.7: Results of the Type III test of fixed effects for each variable and factor under the CDD paradigm. NA=Not Applicable

CDD	Development Approach (DA)	Experience (Exp)	Period	Sequence
<i>Correctness</i>	F=31.717; p=<.001	F=8.651; p=0.005	F=2.251; p=0.14	F=1.128; p=0.294
<i>Efficiency</i>	F=42.527; p=<.001	F=1.037; p=0.314	NA	F=21.086; p=<.001
<i>PEOU</i>	F=20.293; p=<.001	F=3.545; p=0.066	F=0.223; p=0.639	F=1.304; p=0.259
<i>PU</i>	F=13.719; p=0.001	F=1.482; p=0.229	NA	NA
<i>ITU</i>	F=5.007; p=0.03	F=0.043; p=0.836	NA	F=3.692; p=0.061
	DA*Experience	Exp*Sequence	Exp*Period	
<i>Correctness</i>	F=2.521; p=0.119	F=0.031; p=0.861	F=3.134; p=0.083	
<i>Efficiency</i>	F=0.904; p=0.347	NA	NA	
<i>PEOU</i>	F=2.322; p=0.134	F=1.339; p=0.253	F=0.005; p=0.941	
<i>PU</i>	F=1.972; p=0.166	NA	NA	
<i>ITU</i>	F=0.72; p=0.4	F=3.565; p=0.065	NA	

Table 1.8: Results of the Type III test of fixed effects for each variable and factor under the MDD paradigm. NA=Not Applicable

MDD	Development Approach (DA)	Experience (Exp)	Period	Sequence
<i>Correctness</i>	F=5.448; p=0.028	F=8.531; p=0.007	F=0.574; p=0.456	F=0.737; p=0.399
<i>Efficiency</i>	F=0.177; p=0.678	F=1.932; p=0.177	F=6.379; p=0.019	F=2.908; p=0.101
<i>PEOU</i>	F=1.735; p=0.192	F=2.257; p=0.145	F=9.111; p=0.004	NA
<i>PU</i>	F=0.17; p=0.684	F=6.139; p=0.02	F=3.026; p=0.095	NA
<i>ITU</i>	F=0; p=0.993	F=11.899; p=0.002	F=6.293; p=0.019	NA
	DA*Experience	DA*Period	Exp*Period	Exp*Sequence
<i>Correctness</i>	F=0.501; p=0.486	NA	F=0.031; p=0.863	F=4.34; p=0.048
<i>Efficiency</i>	F=0.754; p=0.394	NA	F=0.37; p=0.549	F=0.632; p=0.435
<i>PEOU</i>	F=0.572; p=0.452	F=0.885; p=0.356	F=0.216; p=0.643	NA
<i>PU</i>	F=0.565; p=0.46	F=0.514; p=0.48	F=0.111; p=0.742	NA
<i>ITU</i>	F=0.515; p=0.48	F=1.493; p=0.233	F=0; p=0.986	NA

$DV \sim DA + Exp + Period + Sequence + DA * Exp + Exp * Period + Exp * Sequence + (1|Subj.)$.

The factors and combinations of factors that are part of the selected statistical models explain the changes in the dependent variables; however, according to the test results of the hypotheses, not all the changes in the dependent variables due to these factors were significant. Statistical models such as those in Formula 1.1 and other LMM tested did not verify the normality of the residuals or obtain higher values for the AIC and BIC fit statistics than those obtained by the selected models. With the dataset from the experiment under the CDD paradigm, we obtained normally distributed residuals for Efficiency and PEOU

by using continuous transformations. We used square root transformation for Efficiency, and we used the transformation of squaring for PEOU.

Impact of the Development Approach on Correctness

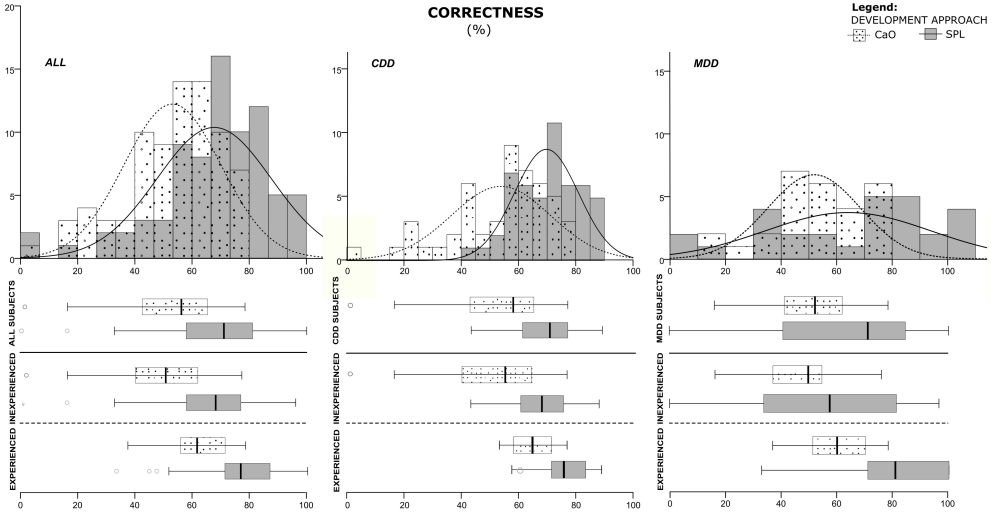


Figure 1.5: Histograms with normal distributions and box plots for Correctness

For Correctness, the DA factor obtained a p-value of less than 0.05 regardless of the paradigm considered. Therefore, we reject the first null hypothesis, and the answer to **RQ1** is affirmative for both paradigms: The DA used for creating a boss in a video game had a significant impact on Correctness regardless of whether the comparison was made under CDD paradigm or under MDD paradigm. In addition, the combination of DA and Paradigm was considered to be statistically significant to explain the changes in Correctness. The bosses made using the SPL were more correct than those made using CaO regardless of whether the comparison was performed under the CDD paradigm or under the MDD paradigm. In addition, the benefits for Correctness of using SPL over using CaO were larger under the CDD paradigm than under the MDD paradigm. The results of the hypothesis tests confirm the statistical significance of the differences observed in the mean values of Correctness for the alternatives of the combination of DA and Paradigm: CDD-CaO (53.87 ± 17.28), CDD-SPL (69.78 ± 11.21), MDD-CaO (51.87 ± 16.64), and MDD-SPL (64.13 ± 29.96).

The effect size of a factor measure through the Cohen d value is related to the percentage of non-overlap between the distributions of the dependent variables

for each alternative of the factor. Higher effect sizes correspond with greater percentages of non-overlap and larger differences. The histograms in Figure 1.5 illustrate the differences in Correctness depending on the DA factor and the combination DA and Paradigm. The first column considers the aggregated data of the two experiments performed, the second column shows the results in Correctness under the CDD paradigm, and the third column shows the results for Correctness under the MDD paradigm. The non-overlapping parts have a single pattern (either dotted or shaded), while the overlapping parts have both patterns (dotted and shaded). In the first histogram, the non-overlapping parts for Correctness are around 30%, which corresponds to large effect size and to a Cohen d value of around 0.8. Under the CDD paradigm, there are more than 40% of non-overlapping parts, which corresponds to a very large effect size and to a Cohen d value higher than 1.0. Under the MDD paradigm, they are around 20%, which corresponds to a medium effect size and to a Cohen d value of around 0.5.

The bottom part of Figure 1.5 shows nine pairs of box plots, arranged in three rows and columns, illustrating the differences in Correctness due to DA. The first row of box plots corresponds to all of the subjects. The second row corresponds to the inexperienced subjects, and the third row corresponds to the experienced subjects. The median of the values of Correctness is higher for SPL than for CaO in all of the box plots and is also higher for experienced subjects than for inexperienced ones. Both the experienced and inexperienced subjects developed more correct bosses when using SPL instead of CaO regardless of the paradigm considered. However, the bosses created by the experienced subjects were more correct than those created by the inexperienced subjects. The Experience factor was considered to be statistically significant to explain the changes in Correctness under both paradigms. However, the changes due to the combination of DA and Experience (DA*Exp) and the combination of Paradigm and Experience (Paradigm*Exp) were not considered to be statistically significant, indicating that the differences in Correctness between the experienced and inexperienced subjects did not depend on either the DA or the Paradigm.

Impact of the Development Approach on Efficiency

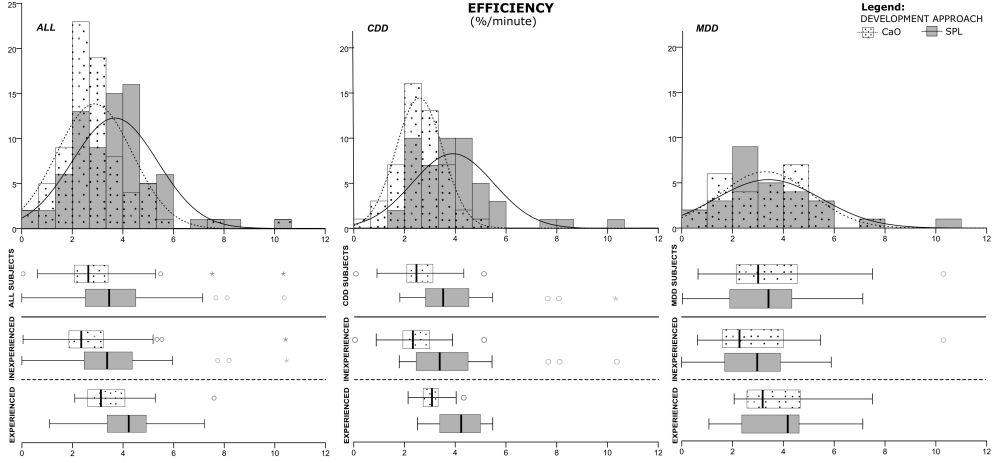


Figure 1.6: Histograms with normal distributions and box plots for Efficiency

For Efficiency, the DA factor obtained a p-value of less than 0.05 for the aggregated data under the CDD paradigm, but greater than 0.05 under the MDD paradigm. Therefore, we cannot reject the second null hypothesis under the MDD paradigm and the answer to the research question **RQ2** is different depending on the paradigm under consideration. Under the CDD paradigm, the DA used had a significant impact on Efficiency and its effect size was large in favor of SPL. In contrast, under the MDD paradigm, the DA used did not have a significant impact on Efficiency. Under the CDD paradigm, the subjects spent less time creating the boss when using the SPL (19.69 ± 5.62 min.) than when using CaO (21.65 ± 4.93 min.). Under MDD paradigm, the subjects spent more time creating the boss when using the SPL (20.6 ± 5.72 min.) than when using CaO (18.43 ± 7.33 min.), which counteracted the positive effect of the use of SPL on Correctness. In addition, the combination of DA and Paradigm was statistically significant to explain the changes in Efficiency. The mean values in Efficiency for the alternatives of these factors indicate that the differences in Efficiency between using the SPL or using CaO were large in favour of the SPL under the CDD paradigm, but were negligible under the MDD paradigm. The results of the hypothesis tests confirm the statistical significance of these differences.

The histograms in Figure 1.6 illustrate the differences in Efficiency depending on the DA factor and the combination DA and Paradigm. The first column considers the aggregated data of the two experiments performed. The second

column shows the results in Efficiency under the CDD paradigm, and the third column shows the results for Efficiency under the MDD paradigm. In the first histogram, the non-overlapping parts for Efficiency are around 20%, which corresponds to a medium effect size. Under the CDD paradigm, they are more than 40%, which corresponds to a very large effect size, and, under the MDD paradigm, they are less than 3%, which corresponds to a negligible effect size.

The factor Experience was considered to be statistically significant to explain the changes in Efficiency. Its effect was large in favor of the experienced subjects. However, the changes due to the combination of DA and Experience (DA*Exp) and the combination of Paradigm and Experience (Paradigm*Exp) on Efficiency were not significant. This indicates that the experienced subjects were more efficient than the inexperienced subjects regardless of the Development approach used (CaO or SPL) or the development paradigm (MDD or CDD). The box plots of Figure 1.6 show the differences in Efficiency due to DA for all of the subjects (the first column), for the subjects participating in the experiment under CDD paradigm (second column), and for the subjects in the experiment under the MDD paradigm (third column). The median of Correctness is higher for SPL than for CaO in all of the box plots. The box plots of Figure 1.6 corresponding to the experienced and inexperienced subjects (the last two rows) show the differences in Correctness due to the combination of Experience and DA.

Impact of the Development Approach on Satisfaction

For the dependent variables related to satisfaction, the DA factor obtained p-values of less than 0.05 under the CDD paradigm but greater than 0.05 under the MDD paradigm. Therefore, we can reject the null hypotheses related to satisfaction only under the CDD paradigm and the answer to research question **RQ3** is different depending on the paradigm under consideration. Under the CDD paradigm, the DA factor had a significant impact in all of the variables related to satisfaction: PEOU, PU, and ITU. In contrast, under the MDD paradigm, DA did not have a significant impact on the satisfaction of the subjects when creating a video game boss. The combination of the DA and Paradigm factors was statistically significant for PU. The subjects perceived that the SPL approach was more useful than the CaO approach, especially under the CDD paradigm.

The Experience factor had significant and large effects on satisfaction only under the MDD paradigm: the experienced subjects scored higher than the inexperienced subjects on PU and ITU for either of the two approaches used

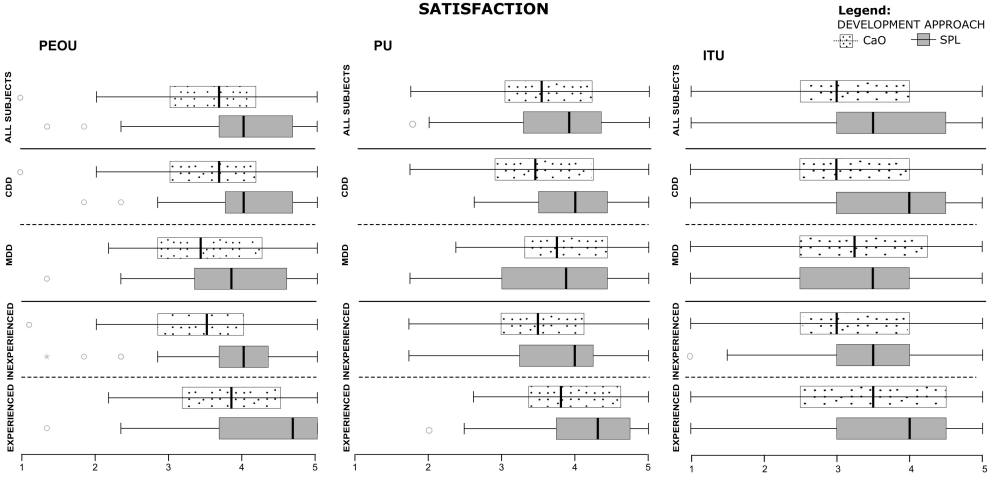


Figure 1.7: Box plots for the dependent variables related to Satisfaction: PEOU, PU, and ITU

in the experiment. However, under the CDD paradigm, the differences in PU and ITU between the experienced and inexperienced subjects were not statistically significant. Under both paradigms, the changes in PEOU due to Experience were not statistically significant.

Figure 1.7 shows fifteen pairs of box plots arranged in five rows and three columns, illustrating the differences in PEOU (the first column), in PU (the second column), and in ITU (the third column) due to DA. The first row of box plots shows the differences for all subjects from both experiments. The second and third rows of box plots show the differences due to the combination of DA and Paradigm. The fourth and fifth rows of box plots show the differences due to the combination of DA and Experience. The values for the median of all of the variables related to satisfaction are higher for SPL than for CaO in all of the box plots, and they are also higher for the experienced subjects than for the inexperienced ones. However, there is hardly any variation between the box plots of the different paradigms for PEOU, while the box plots for PU and ITU show that the differences in favor of SPL are larger under the MDD paradigm than under the CCD paradigm.

Impact of Paradigms on the Differences due to the Development Approach

From the above results, we can answer question **RQ4**. The paradigm under which video game elements were created significantly affected the comparison between the development approaches in terms of Correctness and Efficiency. The differences in favor of using an SPL instead of CaO were greater under the CDD paradigm than under the MDD paradigm, especially in Efficiency. Under CDD there were very large differences in Efficiency in favor of using an SPL, but, under the MDD paradigm, the use of an SPL did not provide significant benefits in Efficiency over the use of CaO. The differences in Satisfaction between the use of SPL and CaO in the two paradigms are not as clear. Although the Satisfaction values in favor of SPL were higher under the CDD paradigm, the differences between the paradigms were only significant in PU, and the differences in all variables related to Satisfaction (PEOU, PU and ITU) in favor of SPL were lower than those obtained in Efficiency or Correction in either of the two paradigms considered.

1.5 Discussion

As reported over the last two decades by the Software Engineering Institute of the Carnegie Mellon University³, SPLs can multiply productivity by a factor of 10, reduce costs up to 60%, reduce labor needs up to 87%, and reduce the time-to-market of new software variants up to 98%. These benefits have been used repeatedly for years to show the attractiveness of SPLs for CSE.

However, our results in the video game domain reveal that efficiency is not always the key aspect that provides SPLs with an advantage over CaO. Existing differences between CSE and GSE (e.g., working on different kinds of artifacts or how they perceive the development process of their projects) may mean that other aspects and uses of SPLs are more relevant for GSE. To better understand the results of the experiments, we carried out focus groups with the subjects.

In addition, the benefits in efficiency of using the SPL instead of CaO were not significant for the subjects in the MDD paradigm, but, under the CDD paradigm, most of the subjects acknowledged that the SPL did accelerate video game development (also shown by our results). When asked about it, the subjects under the MDD paradigm claimed that there was not much difference in working with one model when using CaO (SDML) or the other models

³Available through their online library collection: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=513819>

when using SPL (initial model + variability models); they were similar for the subjects. However, the subjects under the CDD paradigm reported that when using the CaO approach, they had to think in terms of C++, but when working under the MDD paradigm, they were just thinking in terms of the feature model, abstracting away the C++ details. The change between working with SPL or CaO seemed bigger under the CDD paradigm, as they effectively abstracted from code details to focus on the feature model. This may also be augmented by the fact that the SPL used in MDD followed a compositional approach, while the SPL used in CDD used an annotative approach, facilitating the derivation process.

Additionally, most of the subjects observed benefits in SPL for video games, aside from the task requested in the experiments. Specifically, most of the subjects mentioned that the SPL was very relevant for creating new content for video games. As they explained, creating content is critical for video game development. Video game worlds may need a large amount of content to populate them (e.g., non-player characters or items such as weapons or power-ups). Furthermore, there is a trend in the video game industry toward the Games as a Service (GaaS) business model [51]. Once the video game is launched, it is kept alive through updates and additional Downloadable Content (also called DLCs). This new content needs to be cohesive with the already-released game.

The subjects reported that thinking in terms of SPL features helped them think of new content for the video game. The subjects positively valued that they could obtain new content by combining the features following the rules of the variability specification. This new content was different but similar to what already existed in the video game, and the subjects affirmed that this content was relevant to developing video games. The subjects imagine that this new content derived from assembling features can be useful as variants of what was already used in the video game in order to avoid repetition. It could also be used to fill in secondary parts of the game that are more sparsely populated or even as a source of inspiration to create completely new content.

A group of the subjects reported that the SPL may be relevant in helping to balance the difficulty of video games. Balancing the difficulty of video games is one of the main problems of video game development. The video game cannot be too difficult because the players would get frustrated, but it cannot be too easy because the players would lose interest. The subjects thought about how the features could be augmented to include the idea of difficulty in order to derive variants of the same content with different degrees of difficulty.

In the video game research community, content generation for video games is a hot topic. Many surveys [23, 34, 47, 49] talk about how to (completely or partially) automate content generation. Specifically, the automation of content generation is known as Procedural Content Generation (PCG) [3] and uses the most popular Computational Intelligence techniques today (such as machine learning and search-based). However, the SPLs do not appear in any of those surveys. This suggests that the use of SPLs to generate video game content has been neglected by the video game research community and may be an opportunity to explore in the future.

Moreover, we suggest that SPLs need not be viewed solely as an alternative to the current PCG techniques; SPLs can be an ally when combined with PCG techniques. One of the subjects highlighted that the strength of SPL compared to CaO was that only 10 decisions were needed in SPL, while he had to make between 50 and 100 decisions in CaO. This reduced search space of SPLs may be more favorable for the machine learning techniques used in PCG to extract patterns or for the search-based techniques used in PCG to explore the search space.

The experienced subjects acknowledge the benefits of SPLs over CaO under the CDD paradigm. They say SPLs are easier, more accessible, less error-prone, and clearer to understand. However, they sense that they lose expressiveness and control over the software they create. As we have seen in Section 1.4, there are significant differences in favor of SPLs for all of the variables, with the exception of *Intent To Use*. Even though results are favorable toward SPLs, the experienced subjects value source code control of CaO over the many benefits that SPLs offer. Extending the annotated source code was out of the scope of the experiment, but this lack of control would be mitigated in a real scenario, where the developers could also modify the annotated code if needed.

Game development teams are multidisciplinary, and software engineers work together with plenty of non-technical roles (artists, designers, musicians, etc.) that carry the creative part of video games. We claim that the accessibility of SPLs can bring software creation to these roles in a game development team. The engineers focus on building the foundations of the video game with an SPL that comprises the requirements needed, and the other team members can focus on creating content for the game that is new and cohesive with the rest of the content.

One caveat of GSE is that, in general, games have shorter development times than applications in CSE. There is a compromise to be made when building

an SPL system to accelerate development in later development stages. Long developments such as those described in a GaaS context can benefit more from SPLs than short-lifetime video games.

We have identified that the industrial-scale artifact of this work is a game that has already been published. As future work, further research with other video games still in development can bring new perspectives to using SPLs in the video game domain. Finally, one key aspect of video games is that there are many different platforms on which to publish the games. Computers, consoles, and mobiles have plenty of differences that developers need to address when releasing the same game on these platforms. We encourage researchers to explore the benefits of SPLs in this stage of video game development where software reuse and variability are the main artifacts.

1.6 Threats to validity

To describe the threats to validity of our work, we have used the classification of Wohlin et al. [54]:

Conclusion validity is achieved when there is a statistical relationship (with a certain significance) between the treatment and the results. The *low statistical power* was minimized because the confidence interval was 95%. To minimize the *fishing and the error rate* threat, the statistical analysis was done by a researcher who did not participate in the task design or in the correction process. The *Reliability of measures* threat was mitigated because the objective measurements were obtained from the digital artifacts generated by the subjects when they performed the tasks. The *reliability of treatment implementation* threat was alleviated because the procedure was identical in each of the sessions in which the experiment was performed.

Internal validity is achieved when the observed relationships between treatment and outcome are causal relationships and these relationships are not the result of a factor over which we have no control or we have not measured. To avoid the *instrumentation* threat, we conducted a pilot study to verify the design and the instrumentation. The *maturation* threat affected the experiment. Even though the tasks were designed with similar complexity, the effect of period was significant for Efficiency, PEOU, and ITU. The subjects were more efficient in the second task, while PEOU and ITU were better valued in the first task. We minimized this threat by using a crossover design so that the effect affected the evaluated approaches equally. The *selection* threat also affected the experiment because of the voluntary nature of participation. To

avoid student demotivation, we selected students from courses whose contents fit the design of the experiment. The *interactions with selection* threat affected the experiment because there were subjects who had different levels of modeling language knowledge and different levels of knowledge of the video game domain. To mitigate this threat, the treatment was applied randomly and the effects of the sequence factor have been included in the statistical analysis. On the other hand, the number of subjects participating in the experiment under the CDD paradigm was higher than the number of subjects participating under the MDD paradigm, which makes the CDD paradigm more represented in the results associated with the aggregated data. Similarly, inexperienced subjects are more represented in the overall results than experienced subjects.

Construct validity is achieved when the measures actually represent what is being investigated based on the research questions. *Mono-method bias* occurs due to the use of a single type of measure [39]. To mitigate this threat to the correctness and efficiency measurements, we mechanized these measurements as much as possible by means of correction templates. We mitigated the threat to satisfaction measurements by using a widely applied model (TAM) [13]. The *hypothesis guessing* threat appears when the subject thinks about the objective and the results of the experiment. To mitigate this threat, we did not explain the research questions to the subjects. The *evaluation apprehension* threat appears when the subjects are afraid of being evaluated. To weaken this threat, at the beginning of the experiment, the instructor explained to the subjects that the experiment was not a test of their abilities, and in the case of students, that neither their participation nor their results would affect their grades in the course. *Author bias* occurs when the people involved in the process of creating the experiment artifacts subjectively influence the results. In order to mitigate this threat, the tasks were extracted from a commercial video game and were designed by the same expert and researcher with similar difficulty for all of the tasks. These people did not participate in the experiment execution.

External validity is achieved when the results can be generalized outside the settings of the experiment. The *interaction of selection and treatment* threat is an effect of having a subject that is not representative of the population that we want to generalize. The participation of students rather than software engineers is not a major issue as long as the research questions are not specifically focused on experienced developers [28], as is the case in this experiment. In addition, the experience factor has been taken into account in the data analysis. The *interaction of setting and treatment threat* affected the experiment. The focus group revealed that, for both development approaches and paradigms, the subjects spent a lot of time evaluating if the boss being

created was the boss they wanted to develop. The subjects stated that they spent a lot of time testing the video game in real time to understand what they had to develop. This should not be a problem in a real development scenario because it is assumed that the game engineers understand the game they are creating. The *domain* threat occurred because the experiment was conducted in a specific domain (video game) and for a very specific type of task, i.e., to develop a video game boss from Kromaia. We think that the generalization of findings should be undertaken with caution. Other experiments in different games should be performed to validate our findings.

Regarding the generalization of findings, it is important to note the replication limitation of this study. Although we share all the data related to the study results, methodology, and statistical analysis, some of the artifacts related to the infrastructure cannot be disclosed (e.g., game project, DSL, SPL). To mitigate this threat, the experiment was performed using a subset of these systems. We used a subset of the Kormaia content, a subset of the DSL, and a subset of the SPL for the tasks to be understandable and completed in a reasonable amount of time (2 hours). Using these subsets allows both experiments to be performed likewise, using pen and paper, without the need for the complete software infrastructure. However, to evaluate the tasks is needed an experienced researcher in the domain.

All the artifacts needed to perform both experiments are available, along with the complete data set of the results and statistical analysis, in the attached link of Section 1.3.7.

1.7 Related Work

This section presents the related works. It is divided into two subsections taking into account the topics covered in this paper: SPLs applied to the video game domain, and SPL and CaO evaluations or studies.

1.7.1 *SPLs applied to the video game domain*

Some approaches are focused on managing the inherent variability of video games. Boaventura and Sarinho [7] present a collection of game assets to create small video games called Minimal Engine for Digital Games (MEnDiGa). MEnDiGa extends a previous work called FEnDiGa [43], a product line platform that is able to integrate and adapt represented game features in different types of available game engines. In MEnDiGa, they develop logic features and

modules to represent, interpret, and adapt game features for functionality in multiple game platforms. Their results show that the core of game elements developed with MEnDiGa can be independent, reusable, and done in a large-scale way. Castro and Werner [9] present a prototype of a game that was developed by applying a dynamic SPL to generate game modifications systematically. The prototype created demonstrates the possibility of automating the process, having a product line where the original game is the core of the game's functionalities.

Some other research efforts are focused on the creation of SPLs applying re-engineering. Lima et al. [32, 33] present two works about Product Line Architecture (PLA) recovery. They applied their previous proposed approach and guidelines [31] to create the PLA of the Apo-Games project [29]. In addition, they developed an automatic approach for the identification of the minimum subset of cross-product architectural information for an effective PLA recovery. Moreira et al. [38] provide and analyze empirical data of the extraction processes of two open-source case studies, ArgoUML and Phaser. Although the results show that there is a great difference between the re-engineering process of ArgoUML-SPL and Phaser, common problems were found in both cases, which were related to a lack of tools that led to incomplete and inconsistent feature extractions, complexity in managing feature dependencies when using compositional approach, and issues of not having a variability model for dealing with feature constraints.

Similarly, Martinez et al. [35] report an experience concerning the creation of an SPL by re-engineering system variants implemented around an educational game called Robocode. They discuss their results from different perspectives such as educational value, the extractive process, and the analysis of the time and effort required. Debbiche et al. [14] analyze the Apo-Games to identify reusable code or artifacts. They analyzed 5 Java games and migrated 3 of these into a composition-based SPL implemented with *FeatureHouse* [2]. They recommend making sure that the SPL is always testable to ensure the correct transformation to code, and they also recommend incrementally adopting new features to facilitate the extraction.

The above works apply SPL approaches in the video game domain. However, these works do not evaluate whether the benefits produced by the use of SPLs in CSE apply in GSE. In our work, we evaluate the SPL with a commercial video game and with subjects that one linked to the development of video games.

1.7.2 *SPL and CaO evaluations or studies*

Some works focus on comparing different SPL techniques. Constantino et al. [11] compare two SPL tools taking into account ease of use, strengths, and weaknesses. Their results show that the main problem for both tools is the lack of user guides; however, the automated analysis and the feature model editor get very good ratings. Dermerval et al. [15] compared two approaches for feature modeling based on ontologies. Their results suggest that the reasoning capabilities of the ontologies can effectively support product reconfiguration in the context of Dynamics Software Product Lines (DSPLs).

In the context of MDD, Gonzalez-Huerta et al. [22] carried out a family of four experiments to evaluate an MDD method for developing SPLs. They used effectiveness, efficiency, PEOU, PU, and ITU in their comparison. Their experiments show that the method based on MDD obtains the best results.

Some of the research efforts related to CaO focus on features. Ghabach et al. [21] proposed an approach to support the derivation of new product variants based on CaO. Their evaluation of a case study shows that the provided support can considerably reduce the amount of time and effort required to achieve a product derivation. Similarly, Krüger et al. [30] proposed a semi-automatic process to identify and map features between legacy systems. Their results indicate that the process can be used to enable traceability, prepare refactorings, and extract SPLs. Kehrer et al. [27] presented a project called VariantSync to bridge the gap between CaO and SPLs by combining the minimal overhead and flexibility of CaO with the systematic handling of variability in SPLs. They believe that VarianSync has great potential to change the way practitioners develop multi-variant software systems.

After analyzing the above works, we realized that there is a lack of empirical studies comparing SPLs and CaO. Along with the previous paper [50] that this work extends, we were able to find only one paper that performed this comparison [18]. The authors compare the performances of software engineers in the software product development process using SPL and CaO in an industrial context. Their results achieved better values for effectiveness, efficiency, and satisfaction with the SPL approach. In contrast to our work, they focused on CSE and did not address GSE.

1.8 Conclusion

In this paper, we have presented two experiments that evaluate whether the adoption of an SPL in GSE can generate similar benefits as in CSE [18] in terms of correctness, efficiency, and satisfaction. In our experiments, we compared two development approaches, CaO and SPL. Each experiment was conducted under the MDD and CDD paradigms, respectively. A total of 79 subjects participated in the experiments (28 in MDD and 51 in CDD) by developing the final bosses of Kromaia, a commercial video game.

The results show that the bosses developed using the SPL approach were more correct than the bosses developed with CaO under both paradigms. However, there were no significant changes in efficiency when working under the MDD paradigm, while the changes were large and significant when working under the CDD paradigm. Similarly, changes in satisfaction favored SPL rather than CaO and were larger when working under the CDD paradigm than when working under the MDD paradigm. It turns out that the use of SPLs for GSE has a greater impact under the CDD paradigm (where the differences are bigger) than under the MDD paradigm. The subjects of the experiment reported that the change in the way of working with SPL under the CDD paradigm was a real benefit. They could program without code knowledge, focusing only on the feature model and leaving the actual code in the background to the experts.

The results of the previous work [50] revealed that SPLs in GSE can be relevant in generating new video game content (a hot topic in the video game research community) and in balancing the video game difficulty (a seminal problem of video games). This extension supports the relevance of SPLs in GSE, where the differences in favor of SPLs over CaO are greater in a CDD paradigm. Overall, SPLs perform better in terms of correctness, efficiency, and satisfaction under a CDD paradigm. However, the results show that the satisfaction manifested by the subjects with the use of an SPL instead of CaO is not proportional to the benefits that an SPL may have in correction or efficiency, especially in terms of intention to use. We see that developers are still reluctant to use SPLs rather than CaO because of the sense of loss of control over the software they create, even though they perform better when using SPLs.

While SPLs performed better in every aspect than CaO, the SPL approach in GSE did not have the same effects that have made SPLs attractive to CSE. Other applications of SPLs in different games, of different genres, and with different content should be performed to validate the generalization of findings.

There are new trends, such as the GaaS business model, and the particularities of the video game domain that differ from CSE, such as multidisciplinary teams, content creation, difficulty balancing, or the large range of different platforms. These trends and particularities bring novel uses for SPLs within the fertile domain of video games. Therefore, we hope that this work encourages further research of SPLs in GSE.

Acknowledgments

This work has been partially supported by Iniciando Lineas de Producto Software Mediante Busquedas Interactivas Dirigidas por Modelos. Reference: PID2021-128695OB-I00, by Centro para el Desarrollo Tecnológico y la Innovación (CDTI). Reference: SAV-20221011 and in part by the Gobierno de Aragón (Spain) (Research Group S05 20D).

Bibliography

- [1] Apostolos Ampatzoglou and Ioannis Stamelos. “Software engineering research for computer games: A systematic review”. In: *Information and Software Technology* 52.9 (2010), pp. 888 –901. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2010.05.004> (cit. on p. 27).
- [2] Sven Apel, Christian Kästner, and Christian Lengauer. “Language-Independent and Automated Software Composition: The FeatureHouse Experience”. In: *IEEE Transactions on Software Engineering* 39.1 (2013), pp. 63–79. DOI: 10.1109/TSE.2011.120 (cit. on p. 60).
- [3] Nicolas A. Barriga. “A Short Introduction to Procedural Content Generation Algorithms for Videogames”. In: *International Journal on Artificial Intelligence Tools* 28.02 (2019), p. 1930001. DOI: 10.1142/S0218213019300011 (cit. on p. 56).
- [4] Victor R. Basili and H. Dieter Rombach. “The TAME Project: Towards Improvement-Oriented Software Environments”. In: *IEEE Transactions on Software Engineering* (1988). ISSN: 00985589 (cit. on p. 36).
- [5] Daniel Blasco, Carlos Cetina, and Oscar Pastor. “A fine-grained requirement traceability evolutionary algorithm: Kromaia, a commercial video game case study”. In: *Inf. Softw. Technol.* 119 (2020). DOI: 10.1016/j.infsof.2019.106235 (cit. on p. 28).

- [6] Daniel Blasco et al. “An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering”. In: *Journal of Systems and Software* 171 (2021), p. 110804 (cit. on pp. 28, 30, 31, 34).
- [7] Filipe M. B. Boaventura and Victor Travassos Sarinho. “MEnDiGa: A Minimal Engine for Digital Games”. In: *Int. J. Comput. Games Technol.* 2017 (2017), 9626710:1–9626710:13. DOI: 10.1155/2017/9626710 (cit. on p. 59).
- [8] Klara Borowa, Andrzej Zalewski, and Adam Saczko. “Living With Technical Debt—A Perspective From the Video Game Industry”. In: *IEEE Software* 38.06 (2021), pp. 65–70 (cit. on p. 27).
- [9] Diego Castro and Cláudia Werner. “Rebuilding games at runtime”. In: *IEEE*, 2021, pp. 73–77. DOI: 10.1109/ASEW52652.2021.00025 (cit. on p. 60).
- [10] Jacob Cohen. “Statistical power for the social sciences”. In: *Hillsdale, NJ: Laurence Erlbaum and Associates* (1988) (cit. on p. 42).
- [11] Kattiana Constantino et al. “An Empirical Study of Two Software Product Line Tools”. In: *ENASE 2016. SCITEPRESS - Science and Technology Publications, Lda*, 2016, 164–171. ISBN: 9789897581892. DOI: 10.5220/0005829801640171 (cit. on p. 61).
- [12] Crytek. *CRYENGINE / The complete solution for next generation game development by Crytek*. <https://www.cryengine.com>. [Online; accessed 21-November-2021]. 2002 (cit. on p. 27).
- [13] Fred D. Davis. “Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology”. In: *MIS Q.* 13.3 (Sept. 1989), pp. 319–340. ISSN: 0276-7783 (cit. on p. 58).
- [14] Jamel Debbiche et al. “Migrating Java-based Apo-Games into a composition-based software product line”. In: *ACM*, 2019, 18:1–18:5. DOI: 10.1145/3336294.3342361 (cit. on p. 60).
- [15] Diego Dermeval et al. “Ontology-based feature modeling: An empirical study in changing scenarios”. In: *Expert Systems with Applications* 42.11 (2015), pp. 4950–4964. ISSN: 0957-4174. DOI: <https://doi.org/10.1016/j.eswa.2015.02.020> (cit. on p. 61).

-
- [16] África Domingo et al. “Comparing UML-based and DSL-based Modeling from Subjective and Objective Perspectives”. In: Springer. 2021, pp. 483–498 (cit. on pp. 28, 42).
- [17] África Domingo et al. “Evaluating the Benefits of Model-Driven Development - Empirical Evaluation Paper”. In: ed. by Schahram Dustdar et al. Vol. 12127. Lecture Notes in Computer Science. Springer, 2020, pp. 353–367. DOI: 10.1007/978-3-030-49435-3_22 (cit. on p. 28).
- [18] Jorge Echeverría et al. “An empirical study of performance using Clone & Own and Software Product Lines in an industrial context”. In: *Inf. Softw. Technol.* 130 (2021), p. 106444. DOI: 10.1016/j.infsof.2020.106444 (cit. on pp. 61, 62).
- [19] Stefan Fischer et al. “The ECCO tool: Extraction and composition for clone-and-own”. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 2. IEEE. 2015, pp. 665–668 (cit. on p. 28).
- [20] Epic Games. *Unreal Engine: The most powerful real-time 3D creation tool*. <https://www.unrealengine.com>. [Online; accessed 21-November-2021]. 1998 (cit. on p. 27).
- [21] Eddy Ghabach et al. “Clone-and-Own Software Product Derivation Based on Developer Preferences and Cost Estimation”. In: Nantes, France, May 2018 (cit. on p. 61).
- [22] Javier Gonzalez-Huerta et al. “Validating a model-driven software architecture evaluation and improvement method: A family of experiments”. In: *Information and Software Technology* 57 (2015), pp. 405–429. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2014.05.018> (cit. on p. 61).
- [23] Mark Hendrikx et al. “Procedural Content Generation for Games: A Survey”. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 9.1 (Feb. 2013). ISSN: 1551-6857. DOI: 10.1145/2422956.2422957 (cit. on p. 56).
- [24] Evrim Itir Karac, Burak Turhan, and Natalia Juristo. “A Controlled Experiment with Novice Developers on the Impact of Task Description Granularity on Software Quality in Test-Driven Development”. In: *IEEE Transactions on Software Engineering* (2019). ISSN: 0098-5589 (cit. on p. 42).
- [25] Christian Kästner, Sven Apel, and Martin Kuhlemann. “Granularity in software product lines”. In: *30th International Conference on Software*

- Engineering (ICSE 2008)*, Leipzig, Germany, May 10-18, 2008. ACM, 2008, pp. 311–320. DOI: 10.1145/1368088.1368131 (cit. on pp. 34, 35).
- [26] Jussi Kasurinen. “Games as software: Similarities and differences between the implementation projects”. In: *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016*. 2016, pp. 33–40 (cit. on p. 27).
- [27] Timo Kehrer et al. “Bridging the Gap between Clone-and-Own and Software Product Lines”. In: ICSE-NIER ’21. Virtual Event, Spain: IEEE Press, 2021, 21–25. ISBN: 9780738133249. DOI: 10.1109/ICSE-NIER52604.2021.00013 (cit. on p. 61).
- [28] Barbara A. Kitchenham et al. “Preliminary guidelines for empirical research in software engineering”. In: *IEEE Transactions on Software Engineering* 28.8 (Aug. 2002), pp. 721–734. ISSN: 0098-5589 (cit. on p. 58).
- [29] Jacob Krüger et al. “Apo-games: a case study for reverse engineering variability from cloned Java variants”. In: ed. by Thorsten Berger et al. ACM, 2018, pp. 251–256. DOI: 10.1145/3233027.3236403 (cit. on pp. 28, 60).
- [30] Jacob Krüger et al. “Finding Lost Features in Cloned Systems”. In: SPLC ’17. Association for Computing Machinery, 2017, 65–72. ISBN: 9781450351195. DOI: 10.1145/3109729.3109736 (cit. on p. 61).
- [31] Crescencio Lima, Christina Chavez, and Eduardo Santana de Almeida. “Investigating the Recovery of Product Line Architectures: An Approach Proposal”. In: ed. by Goetz Botterweck and Claudia Werner. Cham: Springer International Publishing, 2017, pp. 201–207. ISBN: 978-3-319-56856-0 (cit. on p. 60).
- [32] Crescencio Lima et al. “Recovering the product line architecture of the apo-games”. In: ed. by Thorsten Berger et al. ACM, 2018, pp. 289–293. DOI: 10.1145/3233027.3236398 (cit. on pp. 28, 60).
- [33] Crescencio Lima et al. “Towards an Automated Product Line Architecture Recovery: The Apo-Games Case Study”. In: ACM, 2018, pp. 33–42. DOI: 10.1145/3267183.3267187 (cit. on p. 60).
- [34] Jialin Liu et al. “Deep learning for procedural content generation”. In: *Neural Computing and Applications* 33.1 (Oct. 2020), pp. 19–37. DOI: 10.1007/s00521-020-05383-8 (cit. on p. 56).

-
- [35] Jabier Martinez, Xhevahire Tërnavá, and Tewfik Ziadi. “Software Product Line Extraction from Variability-Rich Systems: The Robocode Case Study”. In: SPLC ’18. Association for Computing Machinery, 2018, 132–142. ISBN: 9781450364645. DOI: 10.1145/3233027.3233038 (cit. on p. 60).
 - [36] Michael McShaffry. *Game Coding Complete*. Paraglyph Publishing, 2003. ISBN: 1932111751 (cit. on p. 27).
 - [37] Daniel L Moody. “The method evaluation model: a theoretical model for validating information systems design methods”. In: *ECIS 2003 proceedings* (2003), p. 79 (cit. on p. 37).
 - [38] Rodrigo André Ferreira Moreira et al. “Open-source software product line extraction processes: the ArgoUML-SPL and Phaser cases”. In: *Empirical Software Engineering* 27.4 (2022) (cit. on p. 60).
 - [39] Jose Ignacio Panach et al. “In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction”. In: *Information and Software Technology* (2015). ISSN: 09505849 (cit. on p. 58).
 - [40] Luca Pascarella et al. “How is video game development different from software development in open source?” In: ed. by Andy Zaidman, Yasutaka Kamei, and Emily Hill. ACM, 2018, pp. 392–402. DOI: 10.1145/3196398.3196418 (cit. on p. 28).
 - [41] Klaus Pohl and Andreas Metzger. “Software Product Lines”. In: ed. by Volker Gruhn and Rüdiger Striemer. Cham: Springer International Publishing, 2018, pp. 185–201. ISBN: 978-3-319-73897-0. DOI: 10.1007/978-3-319-73897-0_11 (cit. on p. 28).
 - [42] Cristiano Politowski et al. “Are game engines software frameworks? a three-perspective study”. In: *Journal of Systems and Software* 171 (2021), p. 110846 (cit. on p. 27).
 - [43] Victor T. Sarinho, Antônio L. Apolinário, and Eduardo S. Almeida. “A Feature-Based Environment for Digital Games”. In: ed. by Marc Herrlich, Rainer Malaka, and Maic Masuch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 518–523. ISBN: 978-3-642-33542-6 (cit. on p. 59).
 - [44] B. Selic. “The pragmatics of model-driven development”. In: *IEEE Software* 20.5 (2003), pp. 19–25 (cit. on p. 27).
 - [45] Martín Sierra et al. “A Comparative Analysis of Game Engines to Develop Core Assets for a Software Product Line of Mini-Games”. In: ed. by

- Xin Peng, Apostolos Ampatzoglou, and Tanmay Bhowmik. Vol. 11602. *Lecture Notes in Computer Science*. Springer, 2019, pp. 64–74. DOI: 10.1007/978-3-030-22888-0_5 (cit. on p. 28).
- [46] SlashData. *Global developer population report 2019*. <https://sdata.me/GlobalDevPop19>. [Online; accessed 21-November-2021]. 2019 (cit. on p. 27).
- [47] Adam James Summerville et al. “Procedural Content Generation via Machine Learning (PCGML)”. In: *IEEE Transactions on Games* 10 (2018), pp. 257–270 (cit. on p. 56).
- [48] Unity Technologies. *Unity Real-Time Development Platform | 3D, 2D VR & AR Engine*. <https://unity.com>. [Online; accessed 21-November-2021]. 2005 (cit. on p. 27).
- [49] Julian Togelius et al. “Search-Based Procedural Content Generation: A Taxonomy and Survey”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 172–186. DOI: 10.1109/TCIAIG.2011.2148116 (cit. on p. 56).
- [50] Jose Ignacio Trasobares et al. “Evaluating the benefits of software product lines in game software engineering”. In: ACM, 2022, pp. 120–130. DOI: 10.1145/3546932.3546998 (cit. on pp. 28, 61, 62).
- [51] Fanny Vaudour and Aleksej Heinze. “Software as a service: Lessons from the video game industry”. In: *Global Business and Organizational Excellence* 39.2 (2020), pp. 31–40 (cit. on p. 55).
- [52] Sira Vegas, Cecilia Apa, and Natalia Juristo. “Crossover designs in software engineering experiments: Benefits and perils”. In: *IEEE Transactions on Software Engineering* 42.2 (2015), pp. 120–135 (cit. on pp. 38, 41).
- [53] Brady T West, Kathleen B Welch, and Andrzej T Galecki. *Linear mixed models: a practical guide using statistical software*. Chapman and Hall/CRC, 2014 (cit. on p. 41).
- [54] Claes Wohlin et al. *Experimentation in software engineering*. Springer Science & Business Media, 2012 (cit. on pp. 28, 29, 36, 39, 57).

Search-based Co-creation of Software Models: The case of particle systems for video games

The video game industry is one of the fastest-growing industries in the world. However, the creation of content is the bottleneck of the industry nowadays. In this paper, we propose a new approach for co-creating content by means of combining an evolutionary algorithm Map-Elites, and software models. Our approach involves generating a large number of software models and selecting the best ones based on a fitness function. This fitness function is guided by the human, who chooses which content fits their interests best. We evaluated this approach in the domain of Particle Systems (PS). PS are a popular type of content used to create visual effects such as explosions, fire, smoke, or rain. Our evaluation also involves industry experts of different roles in the video game development process. Using our approach, they were tasked to create PS for their games. Then, they compared the generated models with handmade ones. Our results show that practitioners chose the generated models four out of five times over handmade ones as a better fit for their projects. Furthermore, models created with our approach by non-experts in five minutes are similar in quality to the ones hand-made by an expert in 15 minutes. In conclusion, using human artistic taste to guide the algorithm renders positive results in creative tasks such as content generation for video games. With minor adjustments, the generated content can be game-ready, accelerating development.

Author's version of the paper: **Jorge Chueca**, Carlos Cetina, Óscar Pastor, Jaime Font.

"Search-based co-creation of software models: The case of particle systems for video games"

Information and Software Technology, vol. 171, p. 107466, Jul. 2024, doi: <https://doi.org/10.1016/j.infsof.2024.107466>.

2.1 Introduction

The video game industry is one of the fastest-growing industries in the world. Almost half of the developers in the world are involved in the video game sector [30]. Most video games are developed using game engines, a development environment that includes the foundations used by most games (such as graphics and physics), and tools to accelerate development. The most popular game engines are Unity [33] and Unreal Engine [12]. They allow video game developers to create content directly using code (e.g., C++ or C#) or software models [38].

Modeling languages, such as UML, are used to create and design software. Game engines use their own Domain-Specific Languages (DSLs) to create certain parts of a video game, following a Model-Driven Engineering (MDE) [17] paradigm. The most notable example is node-based scripting to create shaders: Unreal Engine has Material Editor [23], and Unity Engine has Shader Graph [29].

Nowadays, the generation of content is the bottleneck of the industry. Super productions like *Cyberpunk 2077* can take almost a decade to make [24]. There is a need for more techniques to accelerate the video game creation process, as is the case of Procedural Content Generation (PCG).

However, fully automatic techniques that are traditionally employed for PCG are not always the best for tasks requiring creativity, as is the case of content generation for video games. For instance, search-based techniques optimize a given objective (or set of objectives) iteration after iteration, focusing on a specific area of the search space and sometimes suffering from premature convergence towards local optima [9] and tampering with the creative process.

In this paper, we claim that thanks to the abstraction of software models, it is possible to achieve the co-creation of said models for video game content creation. Software models can represent video game content and be used to

accelerate the creation of content as demonstrated by Blasco et al. [4]. To this end, we propose combining software models with quality diversity algorithms to favor a broad representation of the search space during the search, enabling new, potentially more creative solutions to emerge throughout the search process. The computer performs a search and ensures that the whole search space is represented in the set of solutions. Humans drive the search according to their creative needs, tailoring the process toward the desired result.

Specifically, we combine models from the Meta Object Facility (MOF) [26] with the algorithm Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) [25] as a companion in the creative process. It evolves a population of candidate solutions, assessing them based on a fitness function and evolving them using some genetic operations until the stop condition is met. We use the MAP-Elites algorithm to divide the search space into matrices of cells based on given dimensions of interest. Then, the search process will store the best candidate from each cell (elite) as the population, favoring the diversity of the set of candidates of the population. To this end, we propose an encoder that is able to translate individuals from MOF models to an encoding that can be manipulated by the algorithm, a set of operations to evolve those individuals, and a fitness function based on the feedback given by humans.

As a case study, we use the Particle Systems (PS) domain for video games. PS are present in most video games nowadays, especially 3D games [5]. We created a DSL including the main concepts and relationships of PS that are present in the different commercial video game engines available (Unity, Unreal, etc.). Our approach creates the first generation of individuals based on the model provided as seed. Then, the population of PS is rendered in a simulation so that the human can select favorites for the creative task being performed. The loop is repeated until the stop condition is reached; it can be time-limited or until the user is satisfied with the result.

The case study has been evaluated with video game models created by engineers from a video game company. We compare the created PS of our approach with the handmade PS created by a visual effects expert. Five developers were asked to create five PS to be integrated into one of their games, representing typical desires when creating a video game. A total of 25 PS were created in five collections: *Fire*, *Rain*, *Snow*, *Smoke*, and *Sparks*. The handmade PS is added to each one of these collections. Then, the participants were asked to rank each collection of six PS from the best to the worst.

The results show that game developers tended to prefer co-created PS over handmade PS in four of the five collections of PS. Our results suggest that

this method provides an efficient and effective alternative to the traditional handmade approach. It offers game developers a streamlined and cost-effective way to produce high-quality PS for their games. Additionally, the feedback we received from the focus group suggests that the co-creation approach is particularly accessible to developers with low expertise in visual effects, an essential factor in the continued growth of the game development industry.

The rest of the paper is structured as follows. Section 2.2 presents the motivation and background of the paper. Section 2.3 outlines works related to this study. Section 2.4 presents the proposed co-creation approach. Section 2.5 shows the evaluation performed. Section 2.6 shows the discussion of the results, and, finally, Section 2.8 concludes the paper.

2.2 Background and Motivation

Video games are a mix of art and science. The development of these pieces of software differs from those that tackle traditional Software Engineering problems [2]. Because of the creative nature of video games, developers not only need technical expertise but also artistic expertise, and algorithms for content creation must adapt to those needs.

Search-Based Software Engineering (SBSE) [14] proposes applying a search strategy (such as an evolutionary algorithm) to solve a software engineering problem. To do so, only three key ingredients are needed: (1) a suitable encoding for the problem, which can represent the solution in a format that can be manipulated by the search strategy; (2) a fitness function that can assess how close each candidate is to the solution, guiding the search process; and (3) a set of genetic operators that can evolve individuals of the population, traversing the search space.

However, traditional evolutionary algorithms are limited by the small variety of results that they provide to the human. In a creative process, the developer seeks a wide variety of results in order to ponder the pros and cons of aspects that a machine is not yet capable of understanding; i.e., a computer cannot determine the beauty of a 3D mesh inside a video game or its cohesion in the video game aesthetic. By providing a small number of results, the developer does not have the creative freedom needed. It is desirable to have a large diversity of high-performing, yet qualitatively different solutions, which can be more helpful than a single, high-performing solution.

PS are one of the many content creation tasks that involve creativity when making a video game. The use of PS is one of the key elements that contribute to the visual richness of video games.

To the best of our knowledge, the first time PS were used was in the *Genesis Demo* sequence of the film *Star Trek II: The Wrath of Khan* [28]. PS are visual effects used in video games to simulate various phenomena, such as fire, explosions, smoke, and other environmental effects. These effects are created by multiple small sprites (or 3D meshes), also known as particles, which are controlled by a massive set of parameters such as velocity, size, and color. They are also used to simulate and visualize complex physics, such as fluid dynamics [32].

Commercial game engines such as Unreal Engine and Unity have revolutionized how video games are designed and developed. These game engines use a combination of pre-built and customizable parameters to control the behavior, appearance, and motion of particles in a game scene. Notably, as a case study, we use the PS of the Unity 3D Engine [37], which is a parameter-based system that has different modules that can be added to change the behavior, movement, and appearance of the particles.

Game engines often make use of MDE, as is the case of the Material Editor in Unreal or Shader Graph in Unity. MDE is a software development approach that emphasizes the use of high-level, platform-independent models (PIMs) for designing software systems. These models, expressed in languages like UML or domain-specific languages, allow developers to focus on system structure and requirements without delving into technology specifics. This approach ensures a clear understanding of system design and aligns closely with business goals. The PIM is then automatically transformed into a platform-specific model (PSM), which includes details of a specific technology platform, like a programming language or database system. This step tailors the abstract model to the specifics of the target environment.

In the final stage of MDE, the PSM can be converted into software artifacts (such as source code, or configuration files) compatible with the target platform. This process minimizes human error and maintains consistency between the model and the software product. MDE's automation aspect accelerates development and facilitates quick adaptation to new platforms and technologies. Changes made at the model level can be automatically translated to the code or configuration file, significantly reducing manual coding effort and ensuring efficient platform-specific adjustments. The presented approach can evolve any software model that conforms to a DSL created with MOF, the metalanguage

proposed by the Object Management Group for the creation of DSLs. Those models will be automatically transformed into specific software artifacts (Particle Systems in the case study provided) when needed (e.g. for visualization purposes or as a final export step). The presented approach can be used to create software models and their corresponding software artifacts from other domains (such as 3d meshes) if a suitable DSL that captures the domain is provided.

2.3 Related Work

This section presents other works that are related to this study. It focuses on three aspects of this work: the co-creation of video games, using SBSE for model creation, and putting the human in the loop. We found works addressing these aspects; however, these works do not leverage the characteristics of quality diversity algorithms, or they are not applied to video games, or they do not leverage the generalizability of software models. Our work brings together mixed-initiative model-generation techniques, quality diversity search-based algorithms, and video game content creation. Also, the domain of video games gives great importance to visuals and graphics. The previous works on SBSE and MDE for video game content creation all lack the step in which the developer visualizes and drives the models generated. Our work empowers developers to drive model generation and obtain more favorable solutions in creative domains.

2.3.1 *Co-creation and video games*

There are plenty of works that leverage co-creativity for video games. Lai et al. elaborated a survey on this matter [22]. We focus this section mainly on similar approaches to ours that use the MAP-Elites algorithm or generate PS.

MAP-Elites is an algorithm that has shown promising results in video game co-creation between humans and computers. One of the key strengths of this algorithm is its ability to provide multiple and diverse solutions [25]. This allows for a broader range of creative possibilities and can lead to more interesting and engaging co-created content. As such, MAP-Elites has the potential to enhance the field of video game co-creation significantly. Here, we present examples of previous work in this field.

The MAP-Elites algorithm has been applied in video games to help with level design. For example, Charity et al. created *Baba is Y'all*, a collaborative

mixed-initiative system for building levels for the puzzle game *Baba is You* [7]. The system includes several AI-assisted features to help designers, including a level evolver and an automated player for playtesting. The updated version of the system includes a more user-friendly interface, a better level-evolver and recommendation system, and extended site features [8].

Another work that applies the MAP-Elites algorithm in video games is *Preference-Learning Emitters for Mixed-Initiative Quality-Diversity Algorithms* by Gallotta et al. [11]. Their work uses an interactive constrained MAP-Elites system to learn the designer's preferences and then use them in automated steps. The algorithm was applied to a procedural content generation task creating spaceships in the *Space Engineers* video game.

Content is a broad term that encapsulates a lot of concepts inside a video game. Another case of content is the narrative; Alvarez et al. used the MAP-Elites algorithm to generate diverse and high-performing solutions for game narratives. The algorithm is implemented as a new feature of the Evolutionary Dungeon Designer, a mixed-initiative co-creativity tool for designing dungeons. The feature allows developers to incorporate suggestions produced by the algorithm in the developer's designs. At the same time, any modifications performed by the human will feed back into MAP-Elites, closing a circular workflow of constant mutual inspiration.

Another work that uses autonomous algorithms to generate video game content is by Hastings et al. [15]. Remarkably, the study generates Particle Effects that serve as weapons of spaceships. With the help of Artificial Neural Networks it focuses on generating new content in the form of PS. The authors expanded their work [16] by using the *Galactic Arms Race* game as a case study where players receive new weapons for their spaceships generated by their approach. These weapons are particles created without the control of the developers.

2.3.2 SBSE and software models

None of the aforementioned works leverage the benefits of MDE, directly tackling the complexity of their domains. MDE has proven to be an interesting domain for evolutionary algorithms to thrive. These algorithms can leverage the benefits of software models for the development of video games, where platform independence is a significant problem for game developers who try to publish their games on multiple different hardware.

Williams et al. [35] explored the use of software models and SBSE by using an evolutionary algorithm to identify optimal game character behaviors in a

text-based video game. Character behavior was specified using a Domain-Specific Language. The game's battles were based on text without 3D visuals to represent them.

A recent study [6] proposes a novel way to locate bugs in video games by means of evolving simulations. These simulations mimic player behavior through Non-Playable Characters, producing traces that are relevant to the location of bugs. Their approach has shown promising results in locating bugs in software models of video games.

There is previous work on generating video game content with software models and SBSE. This is the case of Evolutionary Model Generation (EMoGen) by Blasco et al. [4], which generates models of game bosses that are comparable in quality to those created by human developers but in significantly less time. Their approach is based on a traditional evolutionary algorithm without human input.

2.3.3 Humans in the model-generation algorithm

There are many contributions that delve into human intervention in model-generation algorithms. Kessentini et al.[19] propose an interactive multi-objective approach to address the challenges arising from the frequent evolution of meta-models, which often leads to instance models becoming non-conforming to the revised metamodels. The approach dynamically suggests edit operations to developers, aiming to minimize conformance errors, maximize similarity with the initial model, and reduce the number of proposed edit operations while incorporating developers' feedback. This feedback refines the subsequent rankings of recommended edit operations.

Kessentini and Alizadeh [18] present an interactive multi-objective approach to address challenges in metamodel/model co-evolution. The approach dynamically suggests edit operations to designers, focusing on minimizing deviation from the initial model, non-conformities with the revised metamodel, and the number of changes. It clusters recommended co-evolution solutions, allowing users to select a preferred cluster and provide feedback on fewer, more focused solutions, which guides subsequent iterations of the search.

Perez et al.[27] introduce humans to replace the fitness function in SBSE completely. Their SBSE algorithm generates software models that are easier for humans to evaluate rather than evaluating code directly. The study is grounded in the notion that humans, despite not being able to assess millions of solu-

tions like heuristics, can offer valuable insights because of the abstract nature of models.

Bavota et al. [3] explored in their work the introduction of the developer in a genetic algorithm, making it interactable throughout the execution. There are also studies related to interactive algorithms, that is the case of Hall et al. [13] proposing SUMO, an algorithm that allows feeding domain knowledge into a remodularization process. There are other works that leveraged generative algorithms for video development. In their work, Trufano et al. [34] create smart agents with Reinforcement Learning to ease the testing tasks of video games.

2.4 Co-creation Approach

Figure 2.1 shows an overview of the proposed approach. It is divided into two parts, the search process performed by the computer (upper part) and the interactions performed by the human in control of the co-creation process. The search process starts with the human providing an initial model to the encoder. Then, the search will proceed, assessing the fitness of the candidates, evolving them, and updating the MAP-elites based on the dimensions supplied. The human will be able to drive the process, visualize the generated models, configure the fitness function, and configure the dimensions in order to explore the interesting areas of the search space. Additionally, three pseudocodes are presented: Algorithm 1 shows the pseudocode of our Mixed-Initiative approach, Algorithm 2 shows the pseudocode for the genetic algorithm, a modified version of the MAP-Elites algorithm, and Algorithm 3 shows the pseudocode for the fitness calculation.

2.4.1 Co-creation: Search Process

The search process is in charge of traversing the search space, assessing thousands of potential candidate solutions, and filtering out the most promising ones to be presented to the human in charge of the co-creation. In this work, we adapt a MAP-Elites algorithm [25] to work with individuals encoding a MOF model. Whereas a traditional search strategy will evolve a single population of individuals sorted by their fitness value, MAP-elites will evolve a set of populations, arranged by dimensions of interest. This ensures that the areas of interest of the human are explored during the process, and thus, individuals representing those dimensions are presented as solutions, bringing diversity to the process. This results in the need for a fourth ingredient (apart from the

Algorithm 1 Mixed-Initiative MAP-Elites.

```

1:  $i \leftarrow \text{initial\_model}$   $\triangleright$  human provides the initial model
2:  $\mathcal{M} \leftarrow \emptyset$   $\triangleright$  create an empty  $N$ -dimensional MAP of elites  $\mathcal{M}$ 
3:  $i_d \leftarrow \text{dimension\_calculation}(i)$   $\triangleright$  calculate dimension of  $i$  to determine the
   corresponding cell in the MAP  $\mathcal{M}$ 
4:  $\mathcal{M}(i_d) \leftarrow i$   $\triangleright$  store the initial model  $i$  in the corresponding cell of the MAP
    $\mathcal{M}$ 
5:  $\mathcal{F} \leftarrow \emptyset$   $\triangleright$  create an empty set for the fitness calculation
6: while !final_selection do  $\triangleright$  when the human performs the
   final_selection, the algorithm finishes
7:    $\mathcal{M} \leftarrow \text{configure\_dimensions}$   $\triangleright$  when the human changes the
     dimensions, all elements of the MAP are rearranged
8:    $\mathcal{F} \leftarrow \text{configure\_fitness}$   $\triangleright$  when the human modifies the set of fitness
     models, all fitness values are recalculated
9:   evolve_individuals( $n, \mathcal{M}, \mathcal{F}$ )  $\triangleright$  human decides to evolve new
     individuals from current population
10:  visualize_models( $\mathcal{M}$ )  $\triangleright$  models of the MAP are decoded and shown to
     the human
11: end while
12: return model_selected  $\triangleright$  when the human performs the final selection, the
     model is stored

```

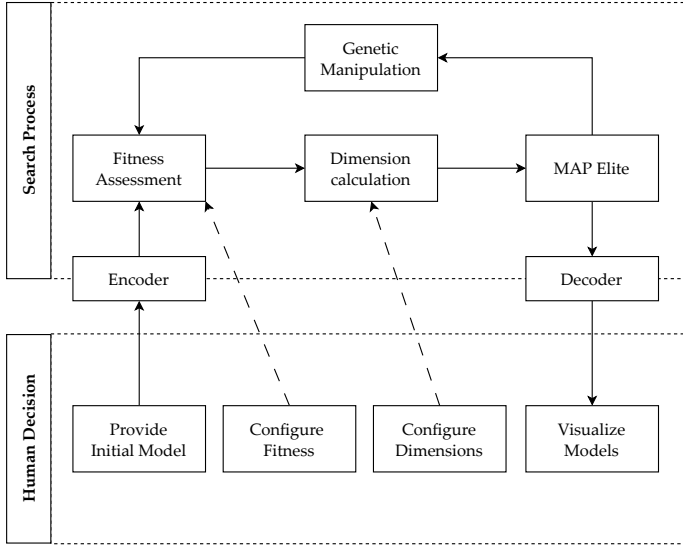


Figure 2.1: Co-Creation approach overview.

encoding, the fitness function, and the genetic operators), i.e., the definition of a set of dimensions of interest of the domain being explored.

By using the MAP-Elites algorithm, we mitigate the premature convergence towards the local optima problem, as the algorithm is evolving a set of populations distributed across the search space. This also fosters diversity of individuals presented to the human, providing means to focus the search on specific dimensions of interests. For example, when evolving particle systems, if one of the dimensions selected is size and quantified as small, medium, or big, the algorithm will evolve three populations in parallel (a population where all members are small, another where all members are medium and another one with all members with big size). As output, the algorithm will provide the best-performing individual of each population, ensuring the representation of each cell of the dimension selected (the best small, the best medium, and the best big).

Algorithm 2 Evolve individuals and store in MAP Elite

```

1: function evolve_individuals( $n, \mathcal{M}, \mathcal{F}$ )  $\triangleright$  number of individuals to evolve
   ( $n$ ), current MAP elite ( $\mathcal{M}$ ) and fitness models ( $\mathcal{F}$ )
2:   for  $1 \rightarrow n$  do
3:      $r \leftarrow \text{random\_selection}(\mathcal{M})$   $\triangleright$  randomly select an individual  $r$  from
        $\mathcal{M}$ 
4:      $x \leftarrow \text{genetic\_manipulation}(r)$   $\triangleright$  create a copy of  $r$  and randomly
       modify a gene
5:      $x_f \leftarrow \text{fitness\_assessment}(x, \mathcal{F})$   $\triangleright$  calculate the fitness of  $x$  and store
       it in  $x_f$ 
6:      $x_d \leftarrow \text{dimension\_calculation}(x)$   $\triangleright$  calculate dimension of  $x$  to
       determine the cell in the MAP  $\mathcal{M}$ 
7:     if  $\mathcal{M}(x_d) = \emptyset$  or  $\mathcal{M}(x_d) \leq x_f$  then  $\triangleright$  if the cell is empty or its
       occupant's fitness is  $\leq x_f$ , then
8:        $\mathcal{M}(x_d) \leftarrow x$   $\triangleright$  store the new individual  $x$  in the appropriate cell
       of the MAP  $\mathcal{M}$ 
9:     end if
10:   end for
11:   return  $\mathcal{M}$   $\triangleright$  returns the map with the new individuals
12: end function

```

Encoder - Decoder

The search process will start with an initial model provided by the human. However, the search process will use the encoder to turn models into individuals that can be effectively manipulated by the search strategy. The encoder will explore the information provided in the metamodel to generate a suitable representation for models of the explored domain. To this end, the encoder will map each element that is present in the initial model to a gene of the individual, as is done in other works from the literature [10]. However, instead of using a binary value for each element, our approach will store the element type as part of the gene. Each type will be managed differently when performing genetic operations and fitness assessments.

In addition, constraints for custom data types will be obtained to reduce the search space. For instance, a metamodel to create web templates that define an attribute to store the width of a given element. The value of the attribute could be of type *float*, but this would lead to elements that are too big for a screen or even negative values. To avoid this, the domain experts creating the

metamodel should embed annotations formalizing their knowledge about the valid ranges.

As a result of this process, an individual that represents the initial model will be generated. Each of the genes will correspond to one of the elements of the initial model, and a range of valid values will also be associated with each of the genes. Whenever the model is needed (e.g., when the human wants to visualize it), the Decoder will perform the opposite operation, turning the genes from the individual back into a model.

Algorithm 3 Fitness Assessment Function

```

1: function fitness_assessment( $x, \mathcal{F}$ )    ▷ the input is the individual to
   evaluate  $x$  and the set of fitness models  $\mathcal{F}$ 
2:    $d_{all} \leftarrow 0$                     ▷ will hold the average distance to all fitness models
3:   for each  $f$  in  $\mathcal{F}$  do
4:      $d_f \leftarrow 0$                   ▷  $d_f$  will hold the distance between  $x$  and the current
   fitness model  $f$ 
5:     for each  $g_f$  in  $f.genes$  and each  $g_x$  in  $x.genes$  do ▷ for each gene  $g_f$ 
   of  $f$  and its equivalent gene  $g_x$  of  $x$ 
6:        $d_f \leftarrow d_f + \text{calculate\_distance}(g_f, g_x)$  ▷ accumulate distances of
   each pair of genes in  $d_f$ 
7:     end for
8:      $d_{all} \leftarrow d_{all} + (d_f / \text{genes.size})$  ▷ Average  $d_f$  and accumulate in  $d_{all}$ 
9:   end for
10:  return  $d_{all} / \mathcal{F.size}$              ▷ Average  $d_{all}$  and return fitness value for  $x$ 
11: end function

```

Fitness Assessment

The fitness function determines the performance of each individual, driving the search for the best-performing individuals. In this work, the fitness function will measure the similitude of the individual with a set of human-selected models (fitness models). By doing so, the human will be able to drive the search toward his/her preferences by choosing the models that exhibit behaviors similar to those desired.

This fitness assessment made by the human narrows the search in an organic way driven by the sole intuition and creativeness of the subject. The humans get to evaluate the model as it evolves under their control as they choose the models that are fed to the fitness function.

To determine the similitude of the individual with each of the fitness models, each of the genes of the individual will be compared to the genes of the fitness models (after turning them back to an individual using the encoder), giving a score that measures the distance between the model and the fitness model. Each gene will score between 1 (exactly the same) and 0 (the biggest distance possible). The value will be calculated differently based on the type of data stored in the gene:

- **EBoolean:** The score will be 1 if both are the same or 0 otherwise.
- **EEnum:** The score will be 1 if both are the same or 0 otherwise. Depending on how the EEnum is created, a notion of distance or order between the different options could exist. As this is not always the case, our fitness assumes no order.
- **EInt:** The score will be 1 minus the distance between the two values, calculated as the absolute value of the difference between them. A value of 1 means that the values are the same, and a value of 0 means that the distance between the values is the biggest possible.
- **EFloat:** The same calculation as EInt.
- **Constrained EFloat:** The score will be calculated as in the EInt case but considering the given boundaries for the value to determine the maximum distance. For example, if the value is constrained to the [1.0-3.0] range, the maximum distance possible will be 2, so the values 1.5 and 2.0 (distance of 0.5) will yield a score of 0.75 ($0.5/2$).
- **Constrained EInt:** The same calculation as Constrained EFloat.

The score of similitude between an individual and a fitness model will be the average score of each gene. Therefore, the fitness value of the individual will be the average similitude value of the individual and all of the fitness models. This will result in a value between 0 and 1 on how similar the individual is to the set of models used as fitness. Note that having several fitness models with different values for a given gene implies that obtaining the highest score for that gene will not be possible.

Dimension Calculation

A traditional evolutionary algorithm maintains a list of the best-performing individuals across generations. However, in MAP-Elites, the population is organized across N dimensions of variation of interest to the human. MAP-Elites will provide a set of high-performing individuals for each point in the space defined by the given dimensions. This will enrich the diversity of the individuals found by the algorithm.

The set of N given dimensions defines a feature space of interest to the human. The dimensions are discretized (based on human preferences or computational resources available), yielding an N -dimensional feature space where each cell corresponds to one of all of the possible combinations of dimensions.

Once the fitness has been assessed, the dimensions for the individual are calculated, and the corresponding cell is retrieved. If the cell is empty, the candidate is stored in that cell. If the cell is already occupied, the fitness values are compared, and the best-performing one is stored in the cell.

Genetic Manipulation

The genetic operators are the mechanism that the search process uses to traverse the search space. An individual (or set of individuals) is selected based on given criteria, with fitness performance being the most common criteria (selection operator). Then, some genes are changed (mutation operator) and/or combined with other individuals (crossover operator) to generate a new offspring. Finally, the newly created offspring is incorporated into the population based on the given criteria (replacement operator).

In the case of the MAP-elites algorithm, the set of operators needed to traverse the search space properly is reduced. Having a set of individuals spanning different dimensions encourages the diversity of the solutions. It avoids common problems such as the local optima (where an individual has to cross a valley of low-performing solutions to become a global optimum). If the dimensions are appropriately selected, the MAP-Elites algorithm is able to explore the search space using only the mutation operator (and a default selection operator and a replacement operator).

First, a random cell is selected (out of all of the cells in the MAP-Elite). The process is repeated if the cell is empty until a cell with an individual is found. Then, the individual from the cell is copied, and the mutation operator is applied to that individual. Our mutation operator needs to act over individuals

encoded as MOF models. It will randomly select a gene and mutate its value based on the type of value stored:

- **EBoolean:** The mutation operator will invert the boolean value.
- **EEnum:** The mutation operator will randomly select an option from the options declared in the metamodel for that EEnum type.
- **EInt:** The mutation operator will assign a random value out of all of the possible values that can be held in that type (int or float).
- **Constrained EInt:** The mutation operator will assign a random value within the boundaries declared for that type in the metamodel.
- **EFloat:** The same as EInt.
- **Constrained EFloat:** The same as Constrained EInt.

The resulting individual will continue the flow of the algorithm, having its fitness assessed and its dimensions calculated, and will be assigned to its corresponding cell (substituting the existing individual if its fitness value is greater).

2.4.2 Co-creation: Human Decision

The human will have different mechanisms to interact with the search process, allowing him/her to effectively drive the search toward their needs. Specifically, the humans can provide an initial model and start the search process, inspect the models of the MAP-Elite, change the dimensions of the MAP-Elite, and modify the models being used as fitness.

Initial model

The first step in the search process will always come from the human side. The human will provide an initial model (conforming to the Domain Specific Language for which models will be co-created) and start the process based on it. Depending on the needs of the human, the search will start with a default model (to create a new model) or with a more complex model (to refine it). For the specific case of PS we use Unity's default PS. Although the user could use PS already present in their game or downloaded from asset databases on the internet.

Depending on the development stage and the intentions of the developer, this initial model can be any PS. However, starting from an already existing PS

inside the scope of the videogame is better for both the developer and the algorithm. The developer has a constant reference of what is being created, and the algorithm can use it for the fitness calculation, always presenting newly generated PS with some degree of similarity to the initial one.

New generations

Then, the human can execute the MAP-Elites algorithm to evolve the individuals and obtain new solutions from the search space. This will trigger the complete loop of genetic operations, fitness calculation, dimension calculation, and MAP-Elite update, resulting in (potentially) new individuals in the population.

The number of iterations of the search that should be executed will depend on the search phase. Initially, the population will only contain the initial model, and the offspring will have more chances of being stored as part of the population. Therefore, the human will want to execute a low number of generations. As the search proceeds and the fitness level of individuals rises, more iterations will be needed to generate offspring that obtain better fitness levels and are stored in the population. In this situation, the human will require a higher number of generations.

Similarly, the number of generations executed could also be determined using different strategies: execute until the population is filled; execute until a given number of new candidates enter the population; execute until the average fitness value is above a given value.

Model visualization

One of the most important mechanisms that the human has for driving the search process is the visualization and inspection of the population of models. This action will require that the individuals are decoded back as models of the DSL being used. The representation of the models will depend on the nature of the domain and the editors or abstract syntax available for the DSL. For example, some models will be visualized using a tree-viewer, while others will include their graphical representation.

In addition, some models will require transformations to be visualized, like transforming them to code using a given Model-to-Text transformation. The representation used should be the one that is useful for the human making the decisions. For instance, if the DSL represents a template for a web page, the

valuable representation for the human could be a preview of the HTML code generated from that model.

As the MAP-elites algorithm stores cells arranged in dimensions, the individuals will be laid out across two axes, one for each dimension (e.g., a matrix of individuals). Depending on the nature of the DSL, the models could need to be visualized using an external platform (such as a web browser in the case of the web page DSL). The approach includes the means to export the models or their transformation in order to be consumed by those external platforms.

Dimension Configuration

If more than two dimensions are provided for a given domain, the human can modify the dimensions that are used to arrange the population. Thus, the human can select which dimension should be represented by each matrix axis. In addition, the visualization platform should be able to provide this information to the MAP-Elites algorithm in order to determine the dimensions to be used while generating the best-performing models (MAP-Elite).

By changing the dimension, the human can efficiently explore the search space (through the individuals generated) and visualize models corresponding to each cell, thereby encouraging the diversity of the results and facilitating the creative process.

Fitness Selection

The fitness function is based on the similitude between the individuals and the set of models that the human selects. Through the fitness selection mechanism, the human can change the models being used as fitness, driving the search in the desired direction. Having a single model as fitness will imply premature convergence, as all of the models will tend to be too similar to the selected one, thus making the steering of the search process impractical. Having the possibility of changing the fitness during the search and pointing to multiple models as individuals will empower the human to drive the search.

A change in the fitness function will invalidate current fitness calculations and will therefore require a recalculation of the fitness values for all of the individuals stored in the MAP-Elite.

2.5 Evaluation

This section presents the evaluation performed to address the following research question regarding the co-creation approach proposed:

RQ: Can the approach be used to refine an initial model into a model that better fits the needs of game developers? If so, is the perceived quality of the refined model better than the perceived quality of the original model?

2.5.1 *Experimental Setup*

In order to answer the research questions, we follow an evaluation where video game developers interact with the proposed approach to refine models.

We developed a DSL for PS, creating a Model-to-Text transformation that converts models into a representation that the visualizing platform can interpret. This creates a visualizing platform for the PS and the definition of a set of nine dimensions of interest for the PS co-creation.

We conducted a quantitative experiment with video game developers. They created variants of a model, and then they evaluated the generated models against handmade models made by an expert. Then, we performed a qualitative evaluation in the form of a focus group.

2.5.2 *Case Study: Particle System domain*

We chose the Particle System (PS) domain as our case study. We deemed this domain to be appropriate because it combines technical and artistic knowledge. It is also generic enough that almost all video games nowadays use PS in their development. This provides our specific case study with value on its own because it is applicable to multiple kinds of developments. PS can also be used outside the visual effects domain, i.e., for simulations.

Particle System Modeling Language (PSML)

The Particle System Modeling Language (PSML) is a DSL that allows the creation of models representing PS. Figure 2.2 shows an excerpt of the PSML metamodel. It describes the following concepts: Main, Emission, Shape, Color over Lifetime, Size over Lifetime, and Renderer.

- **Main.** Describes the starting parameters of the particles (i.e., start color) and general parameters (i.e., looping).
- **Emission.** Describes the number of particles emitted per second.
- **Shape.** Describes the shape of the emission volume and its size. These shapes can be a sphere, a hemisphere, a cone, a box, or a rectangle.
- **Color over Lifetime.** Describes how the color of each particle varies over the time that it is alive.
- **Size over Lifetime.** Describes how the size of each particle varies over the time that it is alive.
- **Renderer.** Describes how the particles are rendered into the screen, selecting the shader that will be executed and the projection matrix that will be used.

In order to evolve the models created and conform to the DSL, each attribute has annotations that limit the range of values. By doing this, the mutations are within reasonable values that allow the human to properly visualize the PS created, i.e., each color channel (RGBA) goes from 0.0 to 1.0.

These effects often simulate organic and pseudo-random behaviors. Developers do not always want to set a fixed parameter for the attributes but rather prefer to choose a random value within a range or along a curve. For example, to simulate a snow effect it is desirable for the flakes to have different sizes. To this end, some attributes are not primitives like *float* but rather a custom type called *MinMaxCurve* that encloses the data needed to provide the developer with more features over the values to be set. Similarly, the attributes related to color are of the type *MinMaxGradient*, enabling the definition of color transitions.

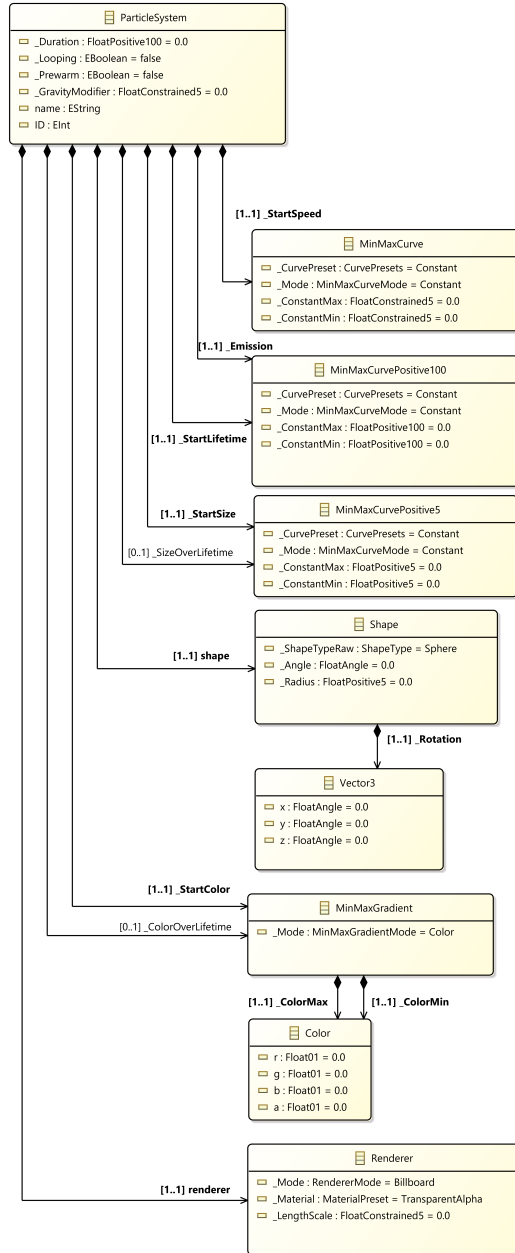


Figure 2.2: Excerpt of the Particle System Modeling Language metamodel.

Dimensions

For the MAP-Elites algorithm to provide a variety of solutions, it needs multiple dimensions to showcase the matrix of solutions. In our case, the matrix is two-dimensional, and the human can choose which dimensions to use for each matrix axis. The list of nine dimensions provided to the human is the following:

- **ColorHueDimension.** This dimension provides a variety of colors to the human by calculating the hue of the color from the RGB values. The hue of a color is represented in degrees ranging from 0 to 360 and is calculated with the following formula:

$$\begin{aligned}
 C_{max} &= \max(R, G, B) \\
 C_{min} &= \min(R, G, B) \\
 \Delta &= C_{max} - C_{min} \\
 H &= \begin{cases} 0^\circ & , \Delta = 0 \\ 60^\circ \times (\frac{G-B}{\Delta} \bmod 6) & , C_{max} = R \\ 60^\circ \times (\frac{B-R}{\Delta} + 2) & , C_{max} = G \\ 60^\circ \times (\frac{R-G}{\Delta} + 4) & , C_{max} = B \end{cases}
 \end{aligned}$$

- **ShapeTypeDimension.** This dimension represents the five types of shapes from which the particles are emitted. These are *Sphere*, *Hemisphere*, *Cone*, *Box*, and *Rectangle*. Each one of the shapes is showcased as a single row or column of the matrix.
- **EmissionDimension.** This dimension represents the particles emitted per second. It is not represented as a primitive *float* but as a *MinMax-Curve* data type that can represent a constant *float*, a random value between two constants, or a value along a curve between two constants. If the mode is *Constant*, the dimension is a single *float* ranging from 0 to 100. If the mode is other than *Constant*, the average value between the two constants is selected, again ranging between 0 and 100.
- **GravityDimension.** This dimension lets the human choose among different values on how gravity affects the particles. This value modifies the direction of gravity and ranges from -5 to 5. If the modifier is positive, gravity works as usual, making the particles fall downwards. The higher the modifier, the higher the acceleration. If the modifier is 0, gravity does

not affect the particles. Finally, if the modifier is negative, gravity moves the particles upwards.

- **LifetimeDimension.** This dimension determines the time that the particle is alive before it is destroyed. It has the same calculation as the *EmissionDimension*, ranging between 0 and 100.
- **MaterialDimension.** This dimension represents the appearance of each particle, selecting a different material that dictates how to render the particles. Each one of the materials has two main characteristics: shape (i.e., changes the image used to be rendered) and blending mode (i.e., changes how each particle is rendered). The four shapes are circle, square, triangle, and line. The three blending modes are alpha blending, additive blending, and opaque. There is a total of 12 materials combining each shape with each blending mode. It is represented by an *enum* that is distributed between the rows or columns depending on the value. This value ranges from 0 to 11.
- **SizeDimension.** This dimension showcases a different selection of particle sizes ranging between 0 and 5. The higher the number, the bigger the particle. Similarly to the other dimensions that rely on values with the type *MinMaxCurve*, this dimension is calculated the same way.
- **SpeedDimension.** This dimension distributes the PS along the rows or columns depending on the initial speed of the particles. Values range from -5 to 5, where negative values make the particles start moving backward and positive values make the particles start moving forward. Values equal to 0 make the particles remain still at the start of their lifetime.
- **SpreadDimension.** This dimension represents the distance among particles when instanced inside the emission shape. It ranges between 0 and 5. Higher values make the particles appear at more distant positions between them, and lower values bring the particles together. If the value is 0, the particles start their lifetimes at the exact same point.

Implementation Details

The presented approach has been implemented and released to the public as part of this work. It can be found at https://svit.usj.es/SBSE_Co-creation_PS/. The search strategy has been implemented within the Eclipse Environment, using the Eclipse Modeling Framework to manipulate the models [31], and based on the pseudo-code provided for the MAP-Elites algorithm [25]. This is the

search process part of the co-creation approach. It takes care of the evolution of the models and the transformation of the models to be used later.

The human part of the co-creation approach has been implemented using the Unity video game engine [33], including a visualizer that can consume the model transformations and turn them into PS that the human can visualize. In addition, all of the mechanisms described as part of the approach are included: fitness selection, dimension configuration, and the possibility of evolving the models.

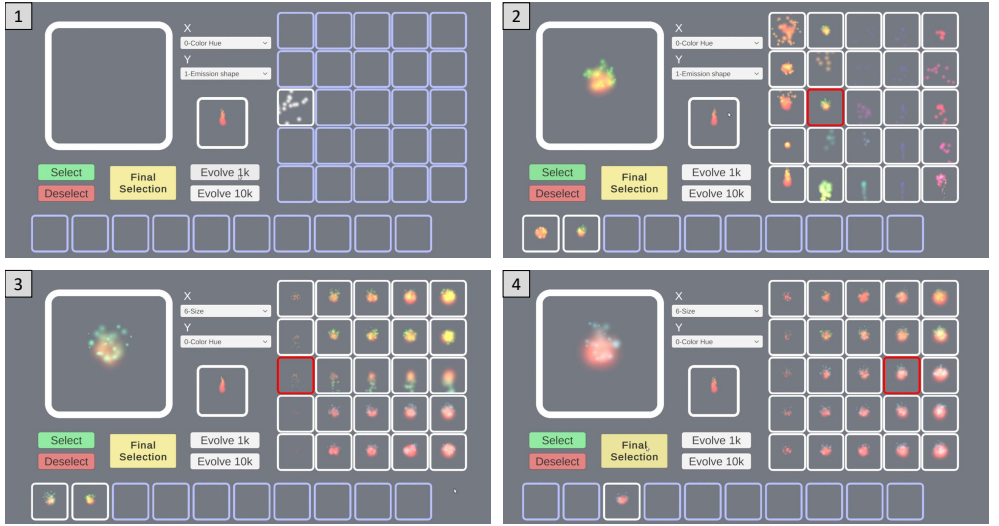


Figure 2.3: Human-part of the Co-Creation Approach for Particle Systems.

Figure 2.3 shows a set of screenshots of our implementation of the human part for the co-creation of PS. It allows the human to visualize each PS created by the search strategy. The population of PS is shown arranged as a matrix in the part of the right. Each axis corresponds to one of the dimensions of interest, which can be selected from a pair of dropdown menus. The lower part shows the set of PS currently selected for the fitness function. On top of the fitness selection, there are controls to add or remove models from the fitness, to finish the search process, and to evolve the models further (either one thousand generations or 10 thousand generations). Finally, the reference PS is in the center.

Specifically, Figure 2.3 shows four screenshots of a co-creation session as seen by the human. The first screenshot shows the initial state of the interface: an empty grid except for one particle system, the initial model that is the default particle system in Unity; an empty line of squares that will show the selected models for the fitness; and the reference PS that is a small fire in this case. The second screenshot shows the state after performing 10k evolutions and then selecting two PS as fitness for the next generations. We can see, in the second screenshot, that the PS are distributed by Color Hue (X-axis) and Emission shape (Y-axis). The use of those dimensions guarantees variety in the set of PS presented to the user, covering the whole spectrum of Color Hue and Emission shape (the dimensions selected). The third screenshot shows the state of the tool after changing the dimensions and performing evolutions with the new selected fitness. We can see, in the third screenshot, that the PS are distributed by Size (X-axis) and Color Hue (Y-axis). Finally, the fourth screenshot shows the final state of the interface after many generations before selecting the desired PS and ending the co-creation process. In this example, the final selection is made after 71k generations in eight different iterations of the algorithm, where each iteration shows 25 different PS to the human, making a total of 200 PS taken into account by the human during the session. The complete session can be seen in the example video from which these screenshots are taken via this link: <https://www.youtube.com/watch?v=vxyGvYp-mYw>.

2.5.3 Results

To evaluate the approach, we conducted an experiment in which five professional video game developers used the tool to create new PS for their games. They belong to the same video game company with considerable heterogeneity of video game genres, and they use PS for all of them. Their games range from a high-speed racing game in VR to a slow-paced point-and-click narrative experience and a physics-based co-op party game. These developments were created using Unity.

First of all, the subjects were asked to fill out a demographic questionnaire. Table 2.1 shows the answers. The Subject column is the subject represented by a letter. The Age column is the age in years. The Dev. The time column is the number of hours the subjects spend developing video games per day. The Programming column is the self-evaluation of their programming knowledge, and the Effects column is a self-evaluation of their PS knowledge, both of which are measured on a 5-point Likert scale.

Subject	Age	Dev. Time	Programming	Effects
A	26	8	4	3
B	30	6	5	4
C	25	7	3	2
D	25	8	4	3
E	27	8	2	1

Table 2.1: Demographic information

The subjects represent a variety of roles in the development process. Two of them are programmers (A, B), one designer (C), one 3D artist (D), and one producer (E). None of them have full proficiency in creating visual effects. While stating somewhat high knowledge of PS, subjects A and B reported that their knowledge is related to the programming of PS without taking into account the artistic part.

After the demographic questionnaire, the subjects were tasked to create five different PS for their games using our approach. The PS to create were *Fire*, *Rain*, *Snow*, *Smoke*, and *Sparks*. They had five minutes per particle system and the population of the matrix started with a single PS, which is the default PS provided by Unity.

A professional visual effects artist has previously hand made the five types of PS in Unity (15 minutes each). These PS were provided to the subjects as a reference for them to create a more refined variant to be used in one of their developments.

After the five subjects finished their tasks, we presented them with five series of six PS (5 subject-created + 1 handmade-created) to be ranked. They needed to order each type by answering the following question: *Which one of the PS is the best fit for one of your developments?* With the first being the best, and the last being the worst.

In order to have a numerical metric from the rankings, we assigned each position a score based on the following formula: $Score = 6 - Position$. This way, the first model selected would have a score of 5, and the last model would have a score of 0. We used a ranking system in the evaluation of our approach to ascertain the feasibility and potential of the proof of concept. It is mostly based on the subjectivity of the taste of each subject and aims to understand the satisfaction with the approach. In this experiment (from the design phase), we ruled out that subjects would use independent values to evaluate the qual-

Chi-square	Fire	Rain	Snow	Smoke	Sparks
Statistic	15.05	13.41	6.14	18.89	20.95
p-value	0.004	0.008	0.09	0.0008	0.0004

Table 2.2: Chi-square statistic and p-values for the rankings of the five PS of the experiment.

ity of the models (e.g. Likert scale or similar) since in cases where subjects perceive similar quality between the models presented, it would not help us to detect whether they prefer the model they have made with the approach to the handmade model (e.g. to indicate that a model is 'comparable' to the handmade one they could give both the same score).

The use of rankings prevents us from applying common analytic approaches such as histograms or analysis of variance [1] as the scores in a ranking are not independent of each other. However, this ranking system allows us to detect when the subjects consider that the model they have made with the approach is better than the handmade model. Furthermore, if the mean values obtained by several models are similar, we can conclude that the differences between them are very small (if I do not know how to classify, I classify at random, then the mean of all models would tend to be the same).

First of all, we will test the rankings provided by the subjects for randomness [1], to determine if there are differences among their preferences expressed through their rankings, or if the differences are mere chance. To do so we have to test the null hypothesis that the mean rank is equal to $\frac{(t+1)}{2}$ for t ranked items. For the six models used in our evaluation, the mean ranking under the null hypothesis of a random ranking would be 3.5 (a score of 2.5 as our score system starts from 0). The test statistic for this null hypothesis is that the mean is distributed as a Chi-square statistic with degrees of freedom of $t-1$.

Table 2.2 shows the results of running the statistic against the null hypothesis for each of the PS effects of the experiment. If the p-value associated with the statistic is smaller than 0.05, we can reject the null hypothesis and conclude that there was a nonrandom pattern in the ranks provided by the subjects. For the snow PS, we cannot reject the null hypothesis, the values provided are too similar to a random ranking, and the subjects' preferences expressed through rankings cannot be distinguished from random chance. We can reject the null hypothesis for the rest of PS: Fire, Rain, Smoke, and Sparks. We can conclude that subjects have uneven preferences over the models, and some are regarded as a better fit than others.

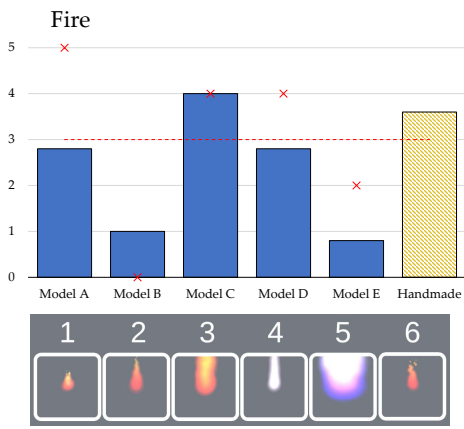


Figure 2.4: Scoring for each *Fire* PS and a screenshot of the ranking video.

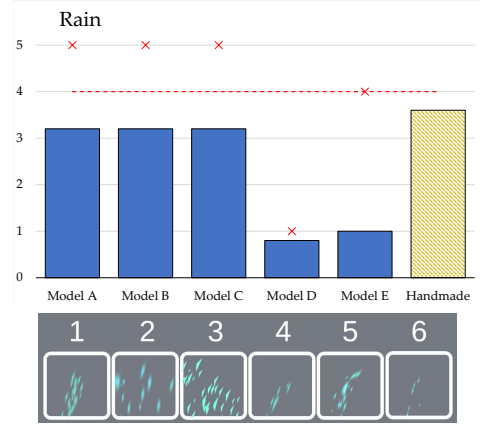


Figure 2.5: Scoring for each *Rain* PS and a screenshot of the ranking video.

The following figures show the average score of each PS as bars and the self-score each developer gives to his/her own model as red Xs. The red dashed line shows the average of the self-scores. Each graph is accompanied by a screenshot of the video presented to each subject for ranking the models. The full videos are accessible via the link in Section 2.5.2.

Figure 2.4, shows the scores for the *Fire* PS. We can see that Model C is ranked higher than the handmade. However, the average self-score was lower than the average score for the handmade PS by 0.6 points. Except Model C the handmade performed better than the other generated models. Additionally, four out of five self-scores are higher or equal to the average score, remarking the difference in perception among subjects when ranking their own models. From these results, we claim that the approach can generate models similar, in satisfaction, to the handmade one.

For *Rain*, Figure 2.5 shows that each subject deemed his/her own PS to be a better fit than the average. Furthermore, on average, they chose their own particle system over the handmade one. Judging by the average of subjects A, B, and C, they performed similarly to the handmade one by only 0.4 points lower each, while the other two generated models performed the worst. While subjects prefer the handmade model in general, they prefer their own models over the handmade one. We can claim that they were satisfied with their own results but not with the results of all the generated models.

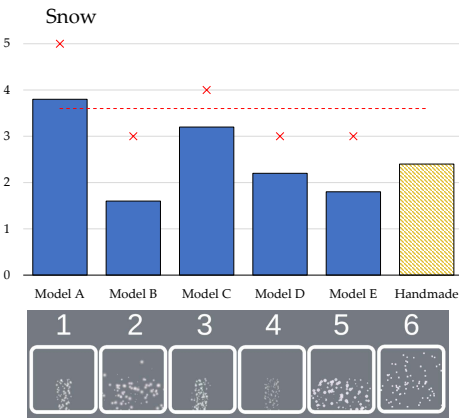


Figure 2.6: Scoring for each *Snow* PS and a screenshot of the ranking video.

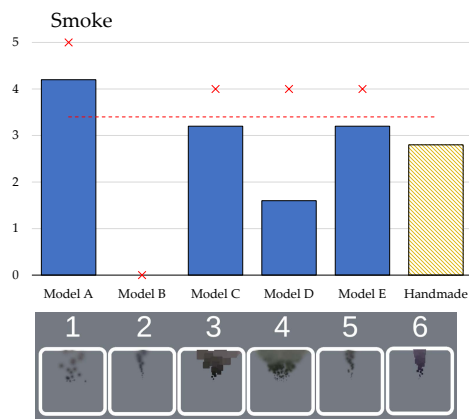


Figure 2.7: Scoring for each *Smoke* PS and a screenshot of the ranking video.

For *Snow*, Figure 2.6 shows that the subjects considered their own models to be better than the handmade one, with over a point of difference. Similarly to *Rain*, all of the subjects considered their own models to be better than the handmade. Two of the models (A and C) are considered better than the handmade on average. We claim that the subjects creating the *Snow* particles were satisfied with their results using the approach.

For *Smoke*, Figure 2.7 shows the best performance for the generated models of all of the PS. The subjects created three models better than the handmade one on average. Also, the self-score was higher than the handmade one (over 0.6 points). In this case, we claim that the subjects were satisfied with their work.

For *Sparks*, the results favor the handmade PS. In Figure 2.8 it can be observed that all of the subjects considered it to be the best fit, and the average self-score was the lowest of them all. However, they still prefer their own generated model rather than the ones generated by other subjects. We claim that the subjects were less satisfied with their work on this type of PS.

Overall, the self-scores were similar to or higher than the handmade score. Also, the subjects preferred generated models to the handmade ones in four out of the five types. The experiment yielded positive outcomes for this proof-of-concept, considering that our approach took only five minutes by non-experts in visual effects, and the handmade ones took 15 minutes by an expert. Our

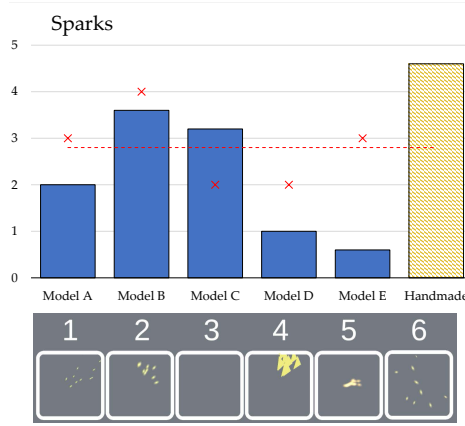


Figure 2.8: Scoring for each *Sparks* PS and a screenshot of the ranking video.

approach is more accessible and fast, obtaining solutions with equivalent in satisfaction to those traditionally made PS.

To conclude, our results show that the subjects consider the models generated by our approach as a similar fit for their games to the one handmade. The exception was the *Sparks*, in which the handmade PS was considered superior. Another key insight that can be derived from these results is the subjectivity of the tasks themselves, where subjects had a focused intention when creating a PS, selecting their own PS 23 out of 25 times as a better fit than the others (the red X is higher than the blue bar).

2.5.4 Focus group

To better understand these results, we conducted a focus group with the subjects in order to obtain their opinions on the tasks and their rationale behind the usage of the approach and the ranks given. To conduct the focus group, we followed the guidelines of Kontio et al. [20] and Krueger [21]. We asked the participants their opinions on the approach with open-ended questions like: “What was your thought process when performing the task?”, “What did you find useful (or not) in this approach?”, or “How would you improve the tool?”. With these questions, we aim to understand what participants think about the approach if they understood its objectives, its flaws, and its strengths. All of the subjects agreed on the usefulness of the approach, highlighting the ease of use and the benefits of the real-time visualization of the PS. We performed the focus group with the same subjects that participated in the experiment.

The duration was 30 minutes, and the discussion themes were related to the satisfaction of the users: the usefulness of the tool, the perceived speed of the development using the approach, and the interface.

In the focus group, the subjects stated that it was difficult to assess the fitness of each PS because the same type of PS can be created for different games with drastically different behaviors. When each subject created a PS, he/she did it with a specific game in mind. For example, one of the *Sparks* can portray a machine malfunction in a video game environment with small bursts of incandescent debris. However, other *Sparks* represent the continuous clash of two metals like a radial saw cutting metal. Participants reported that the task given was broad, and they had to fill a gap in the instruction given (i.e., create a 'rain' for one of your games). Each participant created their own particle system in a particular context, which was open for the interpretation of each participant. Then, in the ranking phase, participants were not looking for the universally best *Sparks*, but the one that better fits the context they added to the task. This context is open to interpretation and differences in opinions and decisions. This thought process that the participants developed is the main reason they were tasked later to rank all the generated PS. Because to evaluate these PS properly, they had to be contextualized. Then, each subject's self-assessment of their models is particularly important. It takes into account how each subject regards their work, and this outcome is supported by the fact that subjects often prefer their own PS over the other ones generated. Ultimately, this approach is destined to be used by the developer, and being able to fulfill the intentions of the human is key for the approach to be successful.

Participants reported that the creation process of a PS with this approach could be divided into two parts we call exploration and refinement. Exploration is the first phase in which participants change dimensions more frequently selecting only a few (from zero to two) PS for the fitness. In this phase, the participant wants to see more variation and thus explore more the domain. The refinement phase begins when the participant is somewhat satisfied with one or two PS. In this second phase, more evolutions are performed without changing dimensions that much, and more PS are selected for the fitness. Finally, the participant selects the final candidate when the algorithm converges into a suitable candidate or when there are only small changes between evolutions.

The subjects had difficulties understanding the concept of dimensions, and they tended only to vary one of the axes while the other stayed fixed for most of the experiment. They also suggested improvements for the user interface and the user experience like accessibility features: *"Would be nice that the color setting had some accessibility features, like telling me the amounts of RGB*

being used” or features improvements like: “*being able to clamp the values of the evolutions*”. Even with the accessibility limitations of the tool, the subjects preferred this approach rather than the traditional slider adjustment.

2.6 Discussion

From our interpretation, the *Sparks* and *Rain* PS presented optimal solutions in the handmade model because those particle systems require high speed to better portray them. As our approach limits some values when using the *Constrained EFloat* data type, we observed that the PS generated for *Sparks* and *Rain* move slower than the one handmade. That is, the search space for some parameters is limited in order to explore the domain efficiently by the approach. This leads to hindered solutions that do not reach the variety of solutions that can be handmade. This limitation can be sorted out if the search is properly guided by the human by selecting the speed dimension to favor the PS with the highest speeds. However, this was unclear for the participants as they reported in the focus group.

This approach is more accessible than learning to use custom tools for each engine. This approach is meant to be a companion for the developer. After generating PS with this approach, the developer can tweak it to their needs. We have seen that the results for the best models created do not necessarily match their assessment of PS knowledge. This suggests that the approach is within everyone’s reach.

Even if this approach is more accessible than the traditional techniques, it is not without its limitations.

In order to accelerate development, some developers buy visual assets from storefronts such as the Unity Store. Our approach will help teams generate unique content with a better fit for their games by creating new content or modifying existing content. The subjects reported that our approach is a good way to prototype and introduce variety in the game. The generation of content is still the biggest bottleneck in the video game industry.

One key point of the approach is the boundaries of the possible values of the attributes. By embedding the domain knowledge into the DSL, we have achieved a reduced search space. This has led to quick and reliable solutions. We can create more boundaries specific to a certain video game context. For example, if the video game uses a specific color palette, the *ColorHueDimension* can be reduced to those colors in the palette, or if the video game is a 2D-pixel

art game, the MaterialDimension can be limited to square-like particles, and the ShapeTypeDimension can be limited to 2D shapes. However, as we have seen in the speed example, these boundaries can limit the search space in a negative way. When using this approach, a balance must be found between domain explorability and solution variety.

These solutions are helpful because of their variety rather than their fitness value. With the MAP-Elites algorithm, we have achieved multiple solutions that the developers can work with, thus, giving them more control over the creative process. The final solutions selected by the subjects did not possess the optimal calculated value by the fitness function. This indicates the significant influence of human decision-making and subjective preferences, suggesting that non-mathematical factors play a critical role in solution selection. This supports the idea of using co-creation algorithms over traditional SBSE ones when applied to creative tasks.

The dimensions used to divide the search space were also beneficial for the search process. When interacting with the approach, the subjects were changing the dimensions based on the specific aspect they wanted to refine; For instance, first obtaining the shape of the PS (using the ShapeType, Material, and Size dimensions) and then switching to ColorHue dimensions to refine the colors and transitions of the effect being co-created. The approach also tackles the main challenges in co-creation: fatigue, lack of control by the human, and little interactivity [22]. Our approach created competitive PS within five minutes because of the human's high level of guidance of the algorithm.

The algorithm is executed over software models that are MOF-compliant, the domain-specific part is Particle Systems that are fed to the human to select and guide the fitness of the algorithm. Although this part is crucial to the execution, any creative domain that can be represented with a MOF meta-model should be usable with this approach. Evaluation with more domains is presented as a future work.

Finally, the approach could be used to iterate and evolve existing particle systems. We can use an example previously made and use it as a selection for fitness from the beginning of the execution of the algorithm. This way, the next generations will tend to be similar to the given model while evolving it providing quality diversity among different dimensions.

2.7 Threats to Validity

To describe the threats to validity of our evaluation, we have used the classification of Wohlin et al. [36]:

External validity is achieved when the results can be generalized outside the settings of the experiment. By applying the algorithm to MOF-compliant models, we ensure that the algorithm can be applied to any DSL that can be visualized and evaluated by a human. The approach is being evaluated in PS content creation and more use cases of the approach have to be evaluated to demonstrate the capabilities of the approach. Moreover, the participants in our experiment are potential users of this tool, which favors the generalization of our results to the industrial environment.

Internal validity is achieved when the observed relationships between treatment and outcome are causal relationships and these relationships are not the result of a factor over which we have no control or we have not measured. The results obtained with the tool may be affected by the parameterizations made in the algorithms used. It is possible that other parameterizations may produce different results. The presented parameters use the experimental parameters terminology from the original MAP-Elites study [25]:

- starting size of the map: 5×5
- final size of the map: 5×5
- genetic operator: mutation
- initial batch: 1
- batch size: 1,000 or 10,000 per iteration, depending on human decision
- number of iterations: human decision
- features (dimensions): 9 showcased in pairs selected by the human
- performance: similarity to human-selected individuals

Construct validity is achieved when the measures actually represent what is being investigated based on the research questions. As it is mentioned before in this paper the fact that humans evaluate the fitness of each model inevitably produces a bias in the findings. However, creating a formula that evaluates the visual appeal of creative content is a subjective task. With the mixed

initiative between machines and humans, the approach makes use of that bias. The quality measure used to classify the models is subjective; each subject in the experiment classifies the models according to his or her criteria and interests. The objective of the approach is to be able to fulfill the creative intentions of the developer.

Conclusion validity is achieved when there is a statistical relationship (with a certain significance) between the treatment and the results. Our experiment is affected by the threat low statistical power, since we have few subjects. To validate our assertion that the tool produces comparable results in quality, a controlled experiment could be performed where different subjects would evaluate the quality of the models presented in this work without knowing their origin (whether they are generated by our approach or handmade) and statistically contrast whether or not they perceive differences in quality.

2.8 Conclusion and future work

Our co-creation approach has proven to work when applied to a real-world scenario, such as the creation of PS for video games. It has been able to support humans in the co-creation of models of the given DSL. Specifically, the fitness function and the configuration options are given to the humans, which can empower them to drive the search toward a model that satisfies their needs.

Five professional video game developers were able to use the approach to refine handmade models into models that better fit their needs in just five minutes. In addition, the developers highlighted the variety of alternatives proposed by the approach, bringing new design ideas that were not part of their initial design but were integrated as part of the final model created.

The proposed approach is generic, and the provided implementation is ready to be applied to co-create models of any MOF-compliant DSL. We expect this to encourage the community to use it and to grow the co-creation of software models. New fitness functions and genetic operators should emerge, empowering humans with efficient ways for driving the search and traversing space towards more exciting paths. Our plans for future works include new validations to determine if the advantages shown in the case of PS can be extended to different creative domains.

Acknowledgements

This work has been partially supported by the following: Iniciando Lineas de Producto Software Mediante Búsquedas Interactivas Dirigidas por Modelos (PID2021-128695OB-I00); Centro para el Desarrollo Tecnológico y la Innovación (CDTI) (SAV-20221011); by the Spanish Ministry of Science and Innovation under the Excellence Network AI4Software (Red2022-134647-T); and in part by the Gobierno de Aragón (Spain) (Research Group S05 20D).

Bibliography

- [1] Mayer Alvo and LH Philip. *Statistical methods for ranking data*. Vol. 1341. Springer, 2014 (cit. on p. 96).
- [2] Apostolos Ampatzoglou and Ioannis Stamelos. “Software engineering research for computer games: A systematic review”. In: *Information and Software Technology* 52.9 (2010), pp. 888–901 (cit. on p. 73).
- [3] Gabriele Bavota et al. “Putting the developer in-the-loop: An interactive GA for software re-modularization”. In: *Search Based Software Engineering: 4th International Symposium, SSBSE 2012, Riva del Garda, Italy, September 28-30, 2012. Proceedings 4*. Springer. 2012, pp. 75–89 (cit. on p. 78).
- [4] Daniel Blasco et al. “An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering”. In: *Journal of Systems and Software* 171 (2021), p. 110804 (cit. on pp. 72, 77).
- [5] John van der Burg. *Building an Advanced Particle System*. <https://www.gamedeveloper.com/programming/building-an-advanced-particle-system>. Accessed: 2023-04-05 (cit. on p. 72).
- [6] Rodrigo Casamayor et al. “Bug localization in game software engineering: evolving simulations to locate bugs in software models of video games”. In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*. 2022, pp. 356–366 (cit. on p. 77).
- [7] Megan Charity, Ahmed Khalifa, and Julian Togelius. “Baba is y’all: Collaborative mixed-initiative level design”. In: *2020 IEEE Conference on Games (CoG)*. IEEE. 2020, pp. 542–549 (cit. on p. 76).

- [8] Megan Charity et al. “Baba is Y’all 2.0: Design and Investigation of a Collaborative Mixed-Initiative System”. In: *IEEE Transactions on Games* (2022) (cit. on p. 76).
- [9] Dario Floreano and Claudio Mattiussi. *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press, 2008. ISBN: 0262062712 (cit. on p. 71).
- [10] Jaime Font et al. “Handling nonconforming individuals in search-based model-driven engineering: nine generic strategies for feature location in the modeling space of the meta-object facility”. In: *Software and Systems Modeling* (2021), pp. 1–36 (cit. on p. 81).
- [11] Roberto Gallotta, Kai Arulkumaran, and LB Soros. “Preference-Learning Emitters for Mixed-Initiative Quality-Diversity Algorithms”. In: *arXiv preprint arXiv:2210.13839* (2022) (cit. on p. 76).
- [12] Epic Games. *Unreal Engine: The most powerful real-time 3D creation tool*. <https://www.unrealengine.com>. [Online; accessed 21-November-2021]. 1998 (cit. on p. 71).
- [13] Mathew Hall, Neil Walkinshaw, and Phil McMinn. “Supervised software modularisation”. In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE. 2012, pp. 472–481 (cit. on p. 78).
- [14] Mark Harman and Bryan F Jones. “Search-based software engineering”. In: *Inf. and soft. Technology* 43.14 (2001), pp. 833–839 (cit. on p. 73).
- [15] Erin J Hastings, Ratan K Guha, and Kenneth O Stanley. “Interactive evolution of particle systems for computer graphics and animation”. In: *IEEE Transactions on Evolutionary Computation* 13.2 (2008), pp. 418–432 (cit. on p. 76).
- [16] Erin Jonathan Hastings, Ratan K Guha, and Kenneth O Stanley. “Automatic content generation in the galactic arms race video game”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 1.4 (2009), pp. 245–263 (cit. on p. 76).
- [17] Stuart Kent. “Model Driven Engineering”. In: *Integrated Formal Methods*. Ed. by Michael Butler, Luigia Petre, and Kaisa Sere. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 286–298. ISBN: 978-3-540-47884-3 (cit. on p. 71).
- [18] Wael Kessentini and Vahid Alizadeh. “Interactive metamodel/model co-evolution using unsupervised learning and multi-objective search”. In:

- Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. 2020, pp. 68–78 (cit. on p. 77).
- [19] Wael Kessentini, Manuel Wimmer, and Houari Sahraoui. “Integrating the designer in-the-loop for metamodel/model co-evolution via interactive computational search”. In: *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. 2018, pp. 101–111 (cit. on p. 77).
 - [20] Jyrki Kontio, Johanna Bragge, and Laura Lehtola. “The focus group method as an empirical tool in software engineering”. In: *Guide to advanced empirical software engineering*. Springer, 2008, pp. 93–116 (cit. on p. 99).
 - [21] Richard A Krueger and Mary Anne Casey. *Designing and conducting focus group interviews*. Vol. 18. Citeseer, 2002 (cit. on p. 99).
 - [22] Gorm Lai, Frederic Fol Leymarie, and William Latham. “On Mixed-Initiative Content Creation for Video Games”. In: *IEEE Transactions on Games* 14.4 (2022), pp. 543–557 (cit. on pp. 75, 102).
 - [23] *Material editor reference, Unreal Engine 4.27 Documentation*. <https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Materials/Editor/>. Accessed: 2023-04-05 (cit. on p. 71).
 - [24] Nick McAdam. *Cyberpunk 2077: 9 years of development, 8 days of hell*. <https://uticatangerine.com/9313/features/cyberpunk-2077-9-years-of-development-8-days-of-hell/>. Accessed: 2023-04-05 (cit. on p. 71).
 - [25] Jean-Baptiste Mouret and Jeff Clune. “Illuminating search spaces by mapping elites”. In: *arXiv preprint arXiv:1504.04909* (2015) (cit. on pp. 72, 75, 78, 92, 103).
 - [26] Object Management Group (OMG). *Meta Object Facility (MOF) Version 2.4.1*. <http://www.omg.org/spec/MOF/2.4.1/>. 2013 (cit. on p. 72).
 - [27] Francisca Pérez et al. “Empowering the human as the fitness function in search-based model-driven engineering”. In: *IEEE Transactions on Software Engineering* 48.11 (2021), pp. 4553–4568 (cit. on p. 77).
 - [28] William T Reeves. “Particle systems—a technique for modeling a class of fuzzy objects”. In: *ACM Transactions On Graphics (TOG)* 2.2 (1983), pp. 91–108 (cit. on p. 74).

- [29] *SHADER GRAPH, Unity Features*. <https://unity.com/features/shader-graph>. Accessed: 2023-04-05 (cit. on p. 71).
- [30] SlashData. *Global developer population report 2019*. <https://sdata.me/GlobalDevPop19>. [Online; accessed 21-November-2021]. 2019 (cit. on p. 71).
- [31] David Steinberg et al. *EMF: Eclipse Modeling Framework 2.0*. 2nd. Addison-Wesley Professional, 2009. ISBN: 0321331885 (cit. on p. 92).
- [32] Jie Tan and XuBo Yang. “Physically-based fluid animation: A survey”. In: *Science in China Series F: Information Sciences* 52.5 (2009), pp. 723–740 (cit. on p. 74).
- [33] Unity Technologies. *Unity Real-Time Development Platform | 3D, 2D VR & AR Engine*. <https://unity.com>. [Online; accessed 21-November-2021]. 2005 (cit. on pp. 71, 93).
- [34] Rosalia Tufano et al. “Using reinforcement learning for load testing of video games”. In: *Proceedings of the 44th International Conference on Software Engineering*. 2022, pp. 2303–2314 (cit. on p. 78).
- [35] James R Williams et al. “Identifying desirable game character behaviours through the application of evolutionary algorithms to model-driven engineering metamodels”. In: *Search Based Software Engineering: Third International Symposium, SSBSE 2011, Szeged, Hungary, September 10-12, 2011. Proceedings 3*. Springer. 2011, pp. 112–126 (cit. on p. 76).
- [36] Claes Wohlin et al. *Experimentation in software engineering*. Springer Science & Business Media, 2012 (cit. on p. 103).
- [37] Bingqing Zhang and Wenfeng Hu. “Game special effect simulation based on particle system of Unity3D”. In: *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*. IEEE. 2017, pp. 595–598 (cit. on p. 74).
- [38] Meng Zhu and Alf Inge Wang. “Model-driven game development: A literature review”. In: *ACM Computing Surveys (CSUR)* 52.6 (2019), pp. 1–32 (cit. on p. 71).

Leveraging Phylogenetics in Software Product Families: The Case of Latent Content Generation in Video Games

A family of software products comprises similar products within a defined scope that share common characteristics, often due to reuse techniques applied during development. This paper introduces an approach that applies biological insights to map the landscape of a software product family, identifying potential gaps within its scope. Phylogenetics studies the gene similarity among groups of organisms to understand ancestry among species. Leveraging Phylogenetics in software, our approach offers a structured view of a product family, aiding in the discovery of unexplored areas fitting the scope of the family. Our approach creates a phylogenetic tree that enables to easily identify latent products (ancestors) that did not exist in the original family. Those ancestors can then be reconstructed from existing products (descendants). The product family evaluated is a set of industry-scale video game non-playable characters. We assess this approach through video game simulations and scope metrics to determine how closely the reconstructed products align with the family's scope. The results confirm that the content generated with phylogenetics aligns better with the family scope than the state-of-the-art procedural content generation techniques using evolutionary algorithms. Phylogenetics enhances content generation by providing a framework to understand and expand the product family with new content.

Author’s version of the paper: **Jorge Chueca**, Daniel Blasco, Carlos Cetina, Jaime Font.

“Leveraging Phylogenetics in Software Product Families: The Case of Latent Content Generation in Video Games”

SPLC24, doi: <https://doi.org/10.1145/3646548.3672596>.

3.1 Introduction

Software is mainly created through reuse. Since the term software engineering was coined at the NATO Conference held in Garmisch in 1968 [42], its evolution has been tied to the concept of reuse (also coined during that conference [38]). Either applying an opportunistic approach [36] such as clone-and-own, or applying a systematic approach as the software product lines propose [13].

These practices result in most products being built by reusing existing ones. Several studies [25, 6, 29, 30] have reported reuse percentages across products ranging from 10% to 85% during the late 80s. More recently, the plastic surgeon theory [4] found that 43% of commits to a large repository of Java projects could be reconstituted from existing code. Similarly, Gabel and Sue [17] concluded that to be able to write a new piece of software in a large repository (sourceforge), an engineer needs to write more than six lines of code; otherwise, the code already exists somewhere.

However, despite extensive reuse in software creation, the increasing demand for software pressures developers who struggle to meet expectations. This scenario is even worse in the case of Game Software Engineering (GSE) [1, 39], where creating new content for video games is the bottleneck in an industry that has become the largest entertainment sector, surpassing music and cinema [44] and accounting for one out of two software developers [54].

To address this need for content, some works from the GSE community have applied Procedural Content Generation (PCG) techniques [22, 56] to accelerate the development of new features for their families of products. However, those techniques are primarily based on evolutionary computation [50], and the generated elements often fall outside the intended scope due to their reliance on randomness. Alternatively, other works use Machine Learning for PCG (PCGML), but the lack of training content remains a challenge for PCGML [49]. Additionally, interpreting the results is challenging for game developers [35], who must create content that meets design expectations.

By contrast, we argue that the inherent reuse across a family of video game elements enables an analysis to generate new content by reusing parts of the family. In this work, we propose using phylogenetics, a nature-inspired approach from biology, to analyze ancestry among family products and exploit them to generate new products within the family scope.

Phylogenetic analysis is used in biology to discover ancestry relationships among individuals, classify them, and formalize an evolution tree that arranges them in a succession of evolutions. This technique yields the discovery of hidden links among the individuals that have yet to be discovered. We argue that this technique can be adapted to a family of software products, yielding the discovery of missing ancestors that can fit into the family of products.

To validate these ideas, we perform a phylogenetic analysis of a family of software products (video game elements from a commercial game), arrange the results into a phylogenetic tree, identify ancestors, reconstruct them using their descendants, and evaluate them as new family products. We then compare these reconstructed ancestors with procedurally generated elements using a state-of-the-art approach, assessing both sets with metrics used in video games to determine the alignment with the game's scope.

The results indicate that the elements generated with our approach align more closely with the original family of products' scope than those generated with the state-of-the-art approach, representing a smaller scope difference 75.88% of the time. The statistical analysis shows that the differences are significant, and there is a large effect size between our approach and the baseline. Applying the proposed technique can create new content and provide a new perspective on the video game's product line.

This study paves a new path in content generation. Previous approaches generate content that is considered secondary content (e.g., vegetation) because they are not aligned with the scope of video games. This is a new point of view for SPLs where genetic divergence points and ancestry branches are exploited. This new perspective opens new possibilities for understanding variability. For instance, Phylogenetics has the potential to initialize an SPL via re-engineering, among other uses.

The paper is structured as follows. Section 2 presents the background for this work. Section 3 presents the steps in our Latent Content Generation approach. Section 4 presents the evaluation. Section 5 answers the research questions and presents a discussion of the results. Section 6 presents the study's threats to validity. Section 7 presents the related work. Section 8 concludes the paper.

3.2 Background

This section introduces Phylogenetic inference and its characteristics borrowed from the Biology field and how it is used to create phylogenetic trees in nature.

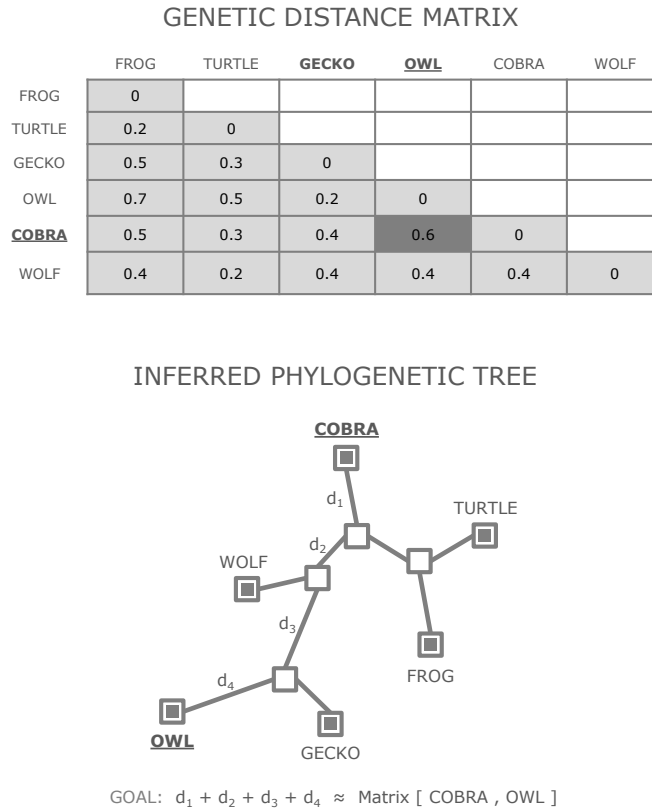


Figure 3.1: Example of an inferred phylogenetic tree from a distance matrix.

In Biology, it is possible to measure the genetic difference between two given species or even between groups of individuals of the same species. That difference is denoted as genetic distance [43], and, in such a field of study, the genetic distance is calculated for two specific taxa. Taxon is a concept that refers to a group of organisms that have a set of genetic characteristics in common. For example, the *Animalia* kingdom is a high-level, general taxon that includes all the animals known, whereas *Canis Lupus* (Grey Wolf) is a low-level taxon that refers to a specific species only. The comparison of a given set of taxa is commonly done using a data structure known as a Genetic Distance Matrix:

a square matrix that represents the genetic distance between each pair of taxa studied [53]. The top part of Figure 3.1 shows how the diagonal values are always zero in a matrix like that. Typically, due to its symmetrical nature, only the distances above or below the main diagonal are represented to hide redundant information. In addition, the genetic distances are usually values between zero and one [23].

It is possible to apply inference techniques to produce a diagram called Phylogenetic Tree [5], which shows how the taxa studied (the leaves of the tree) are related in terms of genetic divergence points and ancestry branches using inner nodes, that hypothesize fossil ancestors, and distance values included in the edges. The lower part of Figure 3.1, which uses an animal species-based example, shows that the objective is to provide additional and useful branching/lineage information while being coherent with the genetic distances used. Therefore, for two given taxa, the sum of the distance values of the path that connects them in the tree should match or be close to the value present in the distance matrix.

3.3 Latent Content Generation

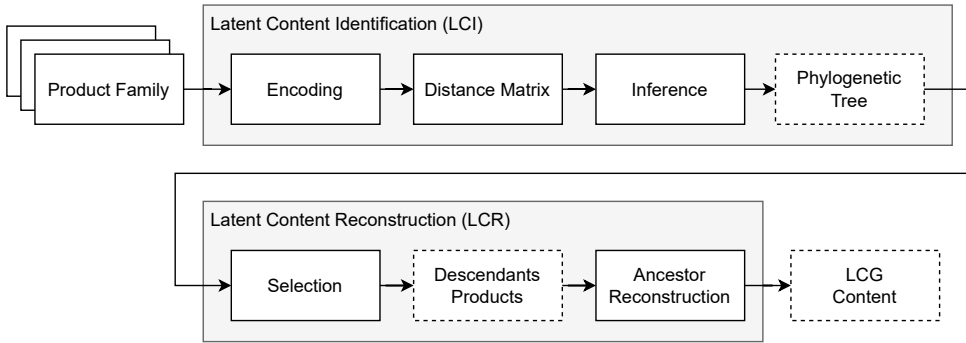


Figure 3.2: Our Latent Content Generation Approach.

Our approach is based on the new perspective that phylogenetics can provide to a product family by representing the genetic relations among products in a single tree. From this tree, the game developer can identify potential gaps (ancestors) that serve as a seed for creating new content. We call this new content as Latent Content.

Figure 3.2 represents an overview of our approach. Dotted boxes represent artifacts generated and solid line boxes represent execution actions or calculations.

The LCG approach is divided into two parts: Latent Content Identification and Latent Content Reconstruction. First, the products in the family are fed into the phylogenetic algorithm to generate the phylogenetic tree. First, each product is encoded into genetic information and then analyzed by pairs to create the distance matrix. To finally create the phylogenetic tree, we perform phylogenetic inference on the collection of taxa, resulting in the visual diagram. Second, With the tree, the developer can select two genetically related products (descendant products) and perform the Latent Content Reconstruction operation to raise the common ancestor between the two products leveraging the information on the commonalities and differences of each descendant.

This approach serves as the first step in the formalization of variability. The generated tree provides an ordered and structured view of the family. We can also create new content that is ensured to be within the family scope by reconstructing common ancestors.

3.3.1 *Encoding*

The first step in the phylogenetic analysis is to encode the content into genetic information. Video game development, a wide range of game content items (e.g., levels/stages or characters) are defined as software models using commercial tools for visual scripting in engines such as Unreal (Blueprints) or Unity (Unity Visual Scripting). A recent survey shows that it is also common to apply Domain Specific Languages (DSLs) [57] for this task. The original DSL used to specify NPCs in Kromaia is SDML. In this work, we apply a string encoding to represent the software models, as others have done for years [7]. It is done by transforming the SDML model into a single string of characters called a genome. Each NPC element (hull, link, weapon, etc.) is represented as a single character in the string. It works similarly to a regular DNA sequence where the order and value of each character are relevant to encode the data.

3.3.2 *Distance Matrix*

When comparing two products, the resulting distance value is stored in a square matrix called distance matrix, as it is commonly used in biology [52]. The values are extracted from 0 (exact copy) to 1 (each gene of the genome is different).

To calculate this distance, we compare each individual’s genome with the other individuals using the Levenshtein distance algorithm [31]. The larger this distance, the smaller the chance that a model is evolved into another model

by a series of changes. However, other distances can be used or created in the future. Each gen is compared one by one, and then the average is computed (i.e., the gene ABA compared with the gene ABC would compute a 0.333).

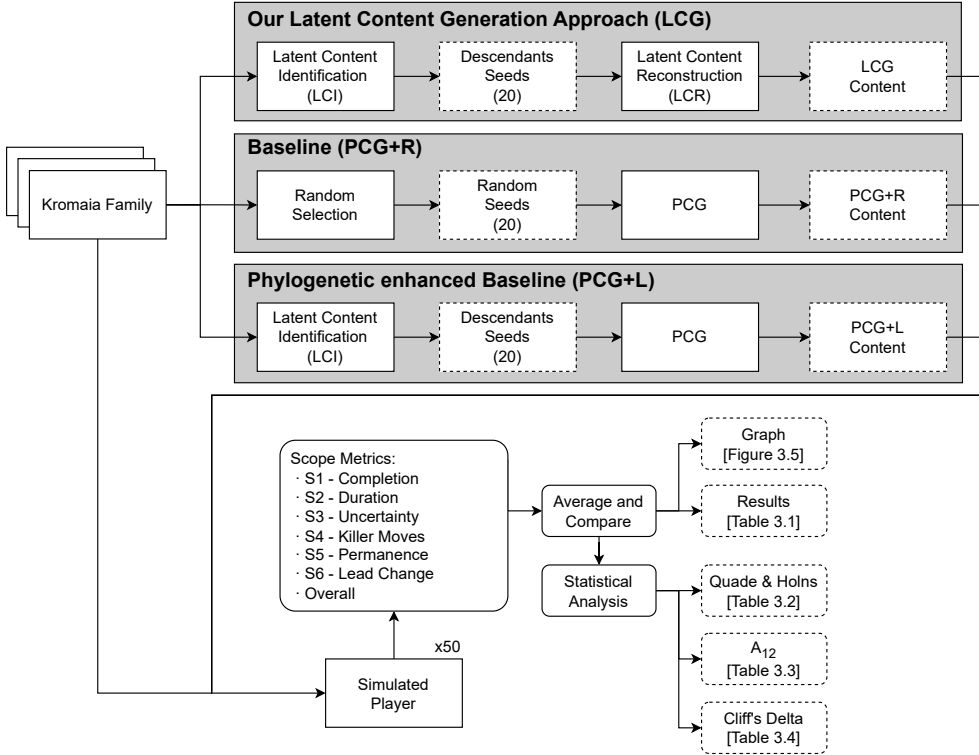


Figure 3.3: Experimental Setup

3.3.3 Phylogenetic Tree Inference

We use the Neighbor-Joining Method from the distance matrix as an inference technique to produce a phylogenetic tree. This method is commonly used in biology phylogenetics [46]. From the genetic distance matrix, each element is compared from the most similar (closer to 0) to the most different (closer to 1). Each comparison creates a new element that replaces the compared elements. These new elements are compared with the other elements using the average of the original values. Here is an example to illustrate the inference calculation. The genetic matrix has the following tuples: $A|B = 0.5$, $A|C = 0.25$, $B|C = 0.33$. The $A|C$ tuple will be processed by replacing A and C with $A^{\wedge}C$. Then,

the tuple $A \hat{C} | B$ is processed with the average of the original values: $A \hat{C} | B = (A | B + C | B) / 2 = 0,415$.

This process is repeated for each pair, storing the creation order to build the tree until only two elements are linked by the root. This root will store the highest value, representing the common genes in all genomes. Figure 3.1 shows an example of a simple phylogenetic inferred from a distance matrix. In that case, the distance $d_1 + d_2 + d_3 + d_4$ should be similar to the distance included in the matrix.

This tree can serve as a visual representation of the product family used as input and the relations among products in terms of genetic similitude. The game developer can spot possible gaps that can be exploited to create new content. These gaps are known in the phylogenetic realm as common ancestors. These ancestors are not present in the original family and serve as hypothetical individuals from which the current individuals could have evolved. We leverage the ancestral data to identify novel hypothetical entities called latent content. While not previously documented, these entities are not entirely unprecedented, embodying characteristics inferred from their descendants.

By inferring the characteristics of newly created individuals from their descendants, we maintain the characteristics of these individuals aligned with the family scope, whether this scope is formalized or not. As video game content is creative by nature, the family scope can sometimes be determined by subjective human requirements such as visual appeal or the fun of the game. Thus, the scope of a family of video game content is not always formalized.

3.3.4 Latent Content Reconstruction

In the phylogenetic tree used in our work, the individuals provided as starting points act as leaves, while the inner, inferred nodes represent points of inflection concerning lineage. When the sibling of a taxon (leaf) is a taxon too and not an inner node, the parent inner node, which is their Most Recent Common Ancestor (MRCA), hypothesizes a taxon from which those two existent ones descend [10]. Our approach uses two taxa with a shared MRCA to reconstruct its genetic material to produce an individual that realizes the latent game content represented by that MRCA.

The reconstruction of that ancestor is done manually, following a set of rules that we defined after studying the information retrieved in interviews with the developers of the video game case study. These criteria take into account the concepts of age and antecedent for elements like files or weapon/projectile

types, referring to the date when they were implemented by the developers or the type that could be considered as the antecedent of the other two given types. The reconstruction rules can be summarized as follows:

- **Anatomy:** The anatomical structure keeps the base infrastructure of the biggest of the two descendants, in terms of spatial organization, but the amount of hull is the average. Additionally, the links that need to be broken due to that new, average organization are removed accordingly.
- **Visual Appearance :** In the case of anatomical characteristics shared by the two descendants, which only differ in the mesh file that gives an object its visual appearance, the oldest of the two visual appearances is kept in the reconstructed ancestor.
- **Weaponry:** The weapon amount is the average of the two descendants, while the weapon types assigned are those considered to be the antecedents of the two when they differ. This criterion is applied to projectile types, too. In both cases, the parameters used could be set by choosing randomly one of the following options: The complete parameter set from the oldest of the two weapon/projectile types and averaged values for every parameter.
- **Behavior:** In the same manner as weaponry, a random decision is made with regard to the use of antecedent types or average parameters.
- **Cosmetic Content:** The objects embedded in the anatomy with purely cosmetic purposes and showing, optionally, animations are only kept if both descendants share them. Otherwise, the reconstructed ancestor will not include those parts.
- **General parameters:** The parameters dealing with characteristics like scale or time values present globally in the descendants in the different aspects described previously are averaged.

These rules represent the first step in consolidating a specific Latent content reconstruction operation. Our purpose is to explore, in the future, possible further installments of that operation, taking into account new perspectives.

3.4 Evaluation

To evaluate our approach, we have designed an evaluation to address the following research questions:

- **RQ1** - Does our Latent Content Generation approach produce results that fit the scope of the family in terms of the metrics used for families of video game content better than the state-of-the-art approach for PCG?
- **RQ2** - If so, how big are the differences between both approaches?
- **RQ3** - Does the Latent Content Identification operation produce better seeds than random for the state-of-the-art approach for PCG?

The state-of-the-art PCG approach is presented in Section 3.4.3.

3.4.1 *Kromaia Product Family*

The main artifact in our evaluation is video game content. Specifically, it is content from *Kromaia*¹, a video game released on both PC and console. This game is distinguished by its dynamic 360-degree movement flight system and shooting mechanics.

The game features Non-Playable Characters (NPCs) that players must overcome. These NPCs vary widely, ranging from small, stationary enemies to large, articulated ones with complex behaviors and weapons. All NPCs align with the developer’s vision, adhering to consistent mechanics, dynamics, and aesthetics [26].

During the development of the game, these NPCs were created using opportunistic reuse by manually copying and pasting parts of some NPCs already created. This opportunistic approach created a product family that has similarities among NPCs. There are common parts that were borrowed from other NPCs, and there are new parts that are particular and unique to each NPC.

These NPCs incorporate concepts common to all entities in the game: hulls, which are rigid bodies in the character’s anatomy, such as beads on a necklace or body parts of a creature; links representing joints with varied degrees of freedom that connect hull pairs; weak points added to regular hulls to define damageable objects; weapons used to attack other characters via projectiles or spikes; and AI components that parametrize behaviors.

¹<https://store.steampowered.com/app/285980/Kromaia/>

3.4.2 Experimental Setup

Figure 3.3 shows the experimental design of the evaluation. The left part shows the family of Kromaia video game content used as input for the evaluation. The family is composed of 48 enemies present in the commercial release of Kromaia. Then, this family is fed as input to the three content generation approaches compared: our Latent Content Generation approach, the state-of-the-art approach for PCG used as a baseline, and a version of the baseline enhanced using our Phylogenetic operation (see grey boxes in Figure 3.3).

Then, the content generated by the three approaches is evaluated using video game simulations and a set of metrics used by the GSE community to assess video game content and guarantee scope alignment (*Completion, Duration, Uncertainty, Killer Moves, Permanence, Lead Change*) [7]. In addition, the original content used in the evaluation is also assessed using those metrics (see dashed arrow). To compare the new content produced with the original content in terms of those metrics to determine if the new content generated is within the scope of the existing content.

Finally, the data is processed to present the results of the evaluation (see right part of Figure 3.3). First, differences among the scope metrics of the new content generated and the original content are computed and presented. Then, a statistical analysis is performed, including measures of statistical significance (Quade and Holm's) [19] and effect size ($\hat{A}_{12}$ and Cliff's Delta) [51, 14, 15].

3.4.3 Content Generation Approaches

To answer the research questions, we compare our LCG approach with the baseline PCG. To determine the PCG Baseline, we had to identify work from PCG literature capable of generating game elements such as the ones of Kromaia. After surveying the literature, we chose the work by Gallota et al. [18] because (1) it is one of the most representative search-based PCG approaches; (2) it generates spaceships for the Space Engineers videogame, so it seemed capable of generating content for Kromaia, and (3) it achieves the best results for this kind of content [18]. The approach from Gallota et al. is a hybrid Evolutionary Algorithm, combining an L-system with a Feasible Infeasible Two Population Evolutionary Algorithm.

Additionally, we explore another way of generating new content, which is the result of combining the operation for selecting the seeds via phylogenetics (the ancestor identification operation) and the PCG evolutionary algorithm. We perform this combination to see how each component affects the results, i.e., we

want to observe how the Latent Content Identification operation can affect the PCG algorithm and, thus, the content generated and how the Latent Content Reconstruction operation affects the results.

As a result, there are three sets of new content generated (see the middle part of Figure 3.3):

- **LCG**: Content created by our Latent Content Generation approach. That is the combination of the seeds obtained with our Ancestor Identification operation followed by the selection of descendants using the phylogenetic tree fed to our ancestor reconstruction operation.
- **PCG+R**: Content created by the baseline PCG approach, fed with random seeds (the usual practice in the literature).
- **PCG+L**: Content created by the baseline PCG approach, fed with the descendant seeds identified by our Ancestor Identification operation, followed by a selection of descendants using the phylogenetic tree.

3.4.4 Game Simulations

The generated content is then evaluated using video game simulations, an accepted practice [11] for video games. Simulations are easy to find in video games, as many of the elements of the game are autonomous (NPCs), and other games directly include contenders (such as racing games).

For the evaluation, we use a simulated player that performs duels against the enemies, simulating what a human player would do during a duel and generating tracing data that will be used to calculate the scope metrics. In the case of Kromaia, the simulated player has been created by the development team, including the configurations needed for their specific intention for the game.

The simulated player is used in our evaluation to execute automatic duels against any type of enemy (the original Kromaia enemies or the generated ones). In those simulations, the simulated player confronts the enemy, strategically moving and targeting the enemy's different hulls and weak points to destroy them. Meanwhile, the enemy will act based on its anatomical structure, behavioral patterns, and attack/defensive dynamics, aiming to destroy the simulated player. Both entities actively strive to emerge victorious, avoiding draws or ties and ensuring a definitive win. As the simulated player is non-deterministic, we will run each duel 50 times to ensure the consistency of

the results across executions. It is no more than 50 times due to time execution constraints. However, it is within the suggested range in the literature [3].

3.4.5 Scope Metrics

Video games are complex software because they interconnect work from various roles (game designers, programmers, 3D artists, or musicians) into a single piece transmitting a pleasant or fun experience to the player. However, 'fun' is an abstract concept and depends on the game designers' intention for their audience. What is fun for some (e.g., a horror movie) can be unpleasant for others. The game designer's mission is to make decisions, take necessary risks, and ensure all elements fit the intended game scope.

Regarding the content of the video game, it must be properly aligned with the experience being evoked to empower game designers to effectively convey their intended experience [48]; mechanics, dynamics, and aesthetics of a video game are interrelated [24], aligned content avoids disrupting the player experience. Therefore, when generating new content for an existing video game (e.g., the family of content from Kromaia), we must ensure that the new content is aligned and remains within the same scope.

There are measurable indicators of game quality in the literature. In this work, we will apply six widely accepted indicators that fit the type of game used in our evaluation (Kromaia, a 3D spaceship shooter) [7, 11]. Our evaluation measures the six scope criteria from 0 to 1 (values will be clamped when needed). Then, we calculate the distance between the generated content and the family of content from Kromaia, obtaining low values for content that is effectively aligned with the scope of the family.

S1-Completion: A duel against an enemy should end with more conclusions (victories for any of the two contenders) than ties (no contender wins, and the duel keeps going for longer than expected by the game designers). Depending on the type of game, duels can be designed to last longer or shorter, so the timeout to consider a duel as a tie can be adapted. In this type of game, ties are abnormal and usually indicate a problem with the content (e.g., cannot be killed by any means). The criterion S1-Completion is the ratio of conclusions over total duels:

$$\text{S1-Completion} = \frac{\text{Conclusions}}{\text{Duels}} \quad (3.1)$$

S2-Duration: The duration of duels between players and enemies is expected to be around an optimal value ($T_{Optimal}$) stated by the game designer. Significant deviations from that reference value are good design-flaw indicators: short games are probably too easy, and long duels tend to make players lose interest. The criterion S2-Duration is the average difference between the duration of each duel (T_d) and the optimal duration ($T_{Optimal}$):

$$\text{S2-Duration} = \text{clamp}_{[0,1]} \left(1 - \frac{\sum_{d=1}^{Duels} \frac{|T_{Optimal} - T_d|}{T_{Optimal}}}{Duels} \right) \quad (3.2)$$

S3-Uncertainty: If a duel outcome can be foreseen in advance, the player might lose interest when she realizes that she will lose the duel and will be bored for the remaining time. To keep players engaged during the duel, the contenders should not get extremely close to victory or defeat too early before the duel finishes. Therefore, a duel is considered to be more uncertain the longer the time until the player's or the enemy's health levels are too low, considered as a dangerous value (P_d and B_d , respectively). For each duel, S3-Uncertainty measures the average deviation between the time at which one of the contenders reaches a dangerous health value and the total duration of the duel (T_d).

$$\text{S3-Uncertainty} = \text{clamp}_{[0,1]} \left(1 - \frac{\sum_{d=1}^{Duels} \frac{T_d - \min(P_d, B_d)}{T_d}}{Duels} \right) \quad (3.3)$$

S4-Killer Moves: While in a duel, contenders perform actions (e.g., moving closer or farther, shooting, using special weapons). Some actions are considered trivial, and others are considered a remarkable highlight (H) towards the duel's outcome. In contrast, others are considered Killer moves (K) as they truly determine the duel's outcome. S4-Killer Moves measures the ratio between killer moves (K) and remarkable highlights (H).

$$\text{S4-Killer Moves} = \text{clamp}_{[0,1]} \left(1 - \frac{\sum_{d=1}^{Duels} \frac{K_d}{H_d}}{Duels} \right) \quad (3.4)$$

S5-Permanence: Permanence measures how often the advantages given by significant actions or moves by one of the contenders are immediately reverted by the opponent in terms of dominance. Recovery moves (R) are those that quickly cancel the advantages gained by the opponent by a killer or highlight move. The criterion S5-Permanence is measured as follows:

$$\text{S5-Permanence} = \text{clamp}_{[0,1]} \left(1 - \frac{\sum_{d=1}^{\text{Duels}} \frac{R_d}{H_d + K_d}}{\text{Duels}} \right) \quad (3.5)$$

S6-Lead Change: The lead of a duel is determined at any given moment by considering the contender with the highest health level. S6-Lead Change measures how often a highlight or killer move changes the lead of the duel (L) as the ratio between changes in the lead (L) and the number of highlight or killer moves (H, K) during the duel:

$$\text{S6-Lead Change} = \text{clamp}_{[0,1]} \left(\frac{\sum_{d=1}^{\text{Duels}} \frac{L_d}{H_d + K_d}}{\text{Duels}} \right) \quad (3.6)$$

Overall Finally, the six scope criteria are combined into a single average value representing the overall scope differences of the evaluated content. This will ease the interpretation of the results:

$$\text{Overall} = \frac{\sum_{i=1}^N S_i}{N} \quad (3.7)$$

3.4.6 Implementation Details

The evaluation has been performed using a commercial HP laptop with an Intel i7-10750H processor, 16Gb of RAM, and Windows 10 64 bits as the host operating system. The execution of the phylogenetic phase took around 10 minutes, with the calculation of the genetic distance matrix being the most time-consuming part (95% of the execution time). This bottleneck happens due to the expensive distance calculation as it needs to iterate for each character

of the NPC files. The presented approach for Latent Content Generation has been implemented using the .NET 6 runtime environment and C# 10 as the programming language.

The scope metrics include configurable parameters tailored to the needs of the game designers. In this work, parameters for those metrics have been provided by the Kromaia development team (determined based on their own experience, desires, or through questionnaires with players) and are as follows: S1-Completion, 20 min. is the maximum time for a duel; S2-Duration, the optimal time for a duel, is 10 min.; S4-Killer Moves, a highlight move happens when either the boss unit or the player experiences a decrease in health, and killer moves are those that make the difference in health between the contenders reach 30%.

The statistical analysis has been performed using R-Studio. The results of the evaluation, the implementation of the approach, the game simulations, and the scripts used for the statistical analysis are made publicly available to the reader².

3.4.7 Results

This section presents the results obtained after applying the content generation approaches.

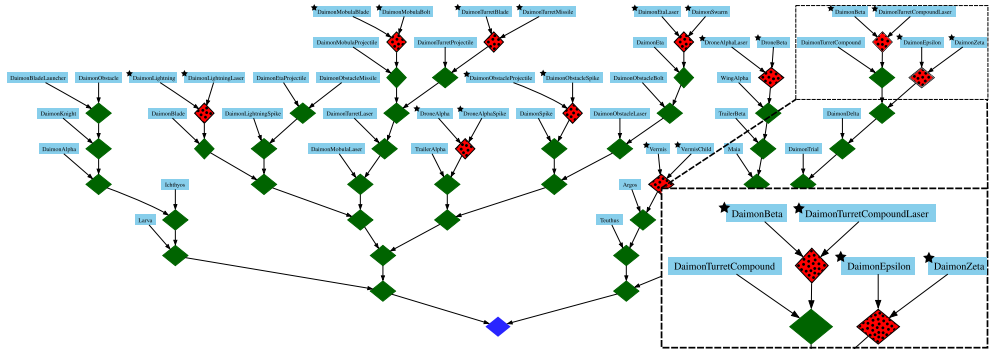


Figure 3.4: Phylogenetic Tree of the family of enemies from Kromaia. Each enemy used as input is depicted as a blue rectangle; ancestors identified are depicted as green diamonds; ancestors reconstructed are depicted as red dotted diamonds.

²<https://anonymous.4open.science/r/LatentContentGeneration-4F32>

	S1 - Completion	S2 - Duration	S3 - Uncertainty	S4 - Killer Moves
Scope	0.99997 ± 0.00015	0.18663 ± 0.01565	0.01313 ± 0.00754	0.82573 ± 0.00232
LCG	0.00003 ± 0	0.07369 ± 0.05910	0.01340 ± 0.00449	0.05175 ± 0.03893
PCG+L	0.00003 ± 0	0.13263 ± 0.06105	0.01792 ± 0.01856	0.14479 ± 0.11813
PCG+R	0.00003 ± 0	0.14780 ± 0.05186	0.02005 ± 0.02252	0.14359 ± 0.12938
	S5 - Permanence	S6 - Lead Change	Overall	
Scope	0.9929 ± 0.0021	0.16053 ± 0.00412	0.52982 ± 0.00399	
LCG	0.00790 ± 0.00881	0.03169 ± 0.4389	0.01122 ± 0.01077	
PCG+L	0.00941 ± 0.00905	0.08366 ± 0.06686	0.02865 ± 0.02102	
PCG+R	0.00954 ± 0.00820	0.08178 ± 0.07408	0.02989 ± 0.02057	

Table 3.1: Reference values for each scope metric and differences for each content generation approach and scope metric. The lower the values the better. The smallest value for each scope metric is highlighted in grey.

The phylogenetic tree generated from the Kromaia product family is presented in Figure 3.4. This tree has a structured view of the product’s family and the genetic relations between products. It serves as the starting point for the reconstruction of latent content. The phylogenetic tree provides descendants from the tree’s leaves (blue rectangles) and its ancestor to be reconstructed (green diamonds). Each ancestor used in the evaluation is represented as a red dotted diamond, and the root of the tree, the common ancestor to all NPCs, is shaded in blue. The selection of ancestry was performed by one professional video game developer of Kromaia with 20 years of experience, and his selection was corroborated by another Kromaia developer. The bottom-right part of Figure 3.4 shows a zoom of two latent ancestors and their corresponding descendants.

As an example, we will use the reconstructed ancestor of the enemies Daimon-MobulaBlade and DaimonMobulaBolt. Both enemies have similar anatomy, visual appearance, and behavior. The main difference is the weaponry. Then, the reconstructed ancestor has the common parts and a random weapon from each enemy with averaged parameters.

The six metrics are obtained by simulations executed for each NPC created and for each product in the Kromaia family. The results show that with LCG, we can create new content more similar to the NPCs in video games. Figure 3.5 and Table 3.1 show the differences for each scope metric between the content generated and the Kromaia family for each approach evaluated. The lower, the better. The differences between our LCG approach and the reference scope

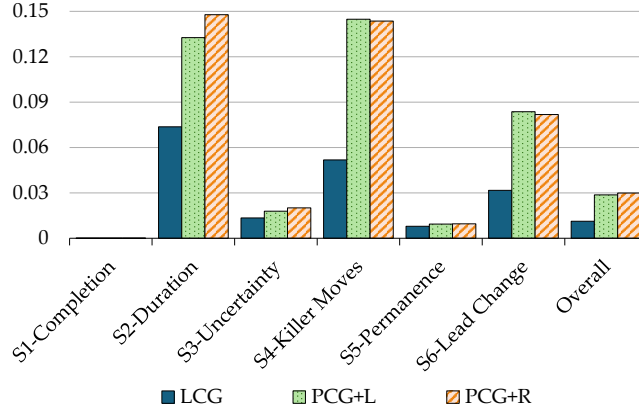


Figure 3.5: Scope differences between the content generated and the family of products from Kromaia. The lower the differences, the better.

metrics from the family (see the first row of Table 3.1) are smaller than those created by the baselines (PCG+R, PCG+L).

We can see that the S1-Completion metric is identical to the scope in the three approaches measured. S1-Completion is the minimum requirement an enemy should meet, for it is expected that the battles, at least, can be completed. Every approach can create content with basic functionality as well as those present in the family. S3-Uncertainty and S5-Permanence are the two other metrics that are more similar to the scope. With these metrics, we can see how LCG starts to outperform the baseline, but the differences seem negligible.

S2-Duration, S4-Killer Moves, and S6-Lead Change are the metrics that show the most difference from the scope. The S2-Duration for LCG is similar to the optimal duration, as indicated by the game designer. S4-Killer Moves and S6-Lead Change show similar performance among approaches, as was expected because these metrics are related with almost three times more in scope in favor of LCG.

All these metrics are summarized with the Overall metric and show that, with LCG, developers can create content that is more similar to the scope than PCG, which is more than twice as different.

3.4.8 Statistical Analysis

To address RQ2, we compare the results obtained from the different content generation approaches. All the data obtained from the simulations were compared to the scope of the family of products and then analyzed following established guidelines [3].

First, a statistical significance analysis will provide formal and quantitative evidence of the differences among approaches, determining whether the differences observed are due to the application of different generation approaches or mere chance. Then, an effect size analysis will be performed to determine if those differences are significant in practice.

Statistical Significance

To determine the statistical significance, we defined two hypotheses: H_0 is the null hypothesis, which states that there are no differences among the content generation approaches; H_1 is the alternative hypothesis, which states that the results of at least one content generation approach differ from another.

Then, a statistical test that returns a probability value (p-value) in the range of 0 to 1 is run. The p-value indicates the probability of the null hypothesis being false (and thus H_0 is rejected, and the alternative hypothesis H_1 is accepted). In this field, a p-value under 0.05 is considered statistically significant.

The statistical test to be applied depends on the nature of the data; in this case, our data does not follow a normal distribution and then requires a non-parametric test. We chose the Quade test, which has shown more power than the rest when applied to a low number of approaches (under 5) and using real data [19].

The first row of Table 3.2 shows the results for the Quade test applied to each scope metric separately. The p-value is below the 0.05 threshold for all metrics except S1-Completion, resulting in an inconclusive test (expected as all values were 0 or almost 0). Therefore, we conclude there are no differences in the first metric (S1-Completion) and differences due to the use of different content generation approaches for the rest of the metrics.

However, the Quade test is only valid to conclude that there are differences but is not able to point out which content generation strategy is better. We need to compare the results of each pair of approaches separately. Holm's post hoc procedure does this, another statistical test commonly used in conjunction

		S1 - Completion	S2 - Duration	S3 - Uncertainty	S4 - Killer Moves
Quade	All	-	$\ll 2x10^{-16}$	$3.0x10^{-13}$	$\ll 2x10^{-16}$
	LCG vs PCG+L	-	$\ll 2x10^{-16}$	0.00014	$\ll 2x10^{-16}$
Holm's	LCG vs PCG+R	-	$\ll 2x10^{-16}$	$1.1x10^{-9}$	$\ll 2x10^{-16}$
	PCG+L vs PCG+R	-	0.12	0.01928	0.059
		S5 - Permanence	S6 - Lead Change	Overall	
Quade	All	$7.0x10^{-8}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	
	LCG vs PCG+L	0.02	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	
Holm's	LCG vs PCG+R	0.02	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	
	PCG+L vs PCG+R	0.99	0.63	0.36	

Table 3.2: Quade test and Holm's Post Hoc p-values. Values below 0.05 are highlighted in grey

with Quade [19]. The test is run for each metric separately, omitting the S1 metric as Quade already showed that differences were not significant.

The second to the fourth row of Table 3.2 shows the p-values of Holm's Post Hoc for each quality metric (columns) and each pair of generation approaches (rows). Values below the threshold (0.05) are highlighted in grey, indicating that the difference in the results for that pair of approaches and scope metric is significant.

We can see that the differences between LCG and the two PCG approaches are significant for the six metrics analyzed in this test (S2, S3, S4, S5, S6, and Overall). According to Holm's test, the differences between both PCG approaches, with latent input or with random input, are not significant except for S3 - Uncertainty.

Effect Size

After concluding that there are differences among the results of the different content generation approaches, we have to determine the effect size of those differences, that is, how big those differences are [21]. Even when having differences, they can be too small and have no practical value (especially when the number of runs is big enough). Therefore, we assess the magnitude of that difference by applying two non-parametric measures, Vargha and Delaney's $\hat{A}_{12}$ [51] and Cliff's Delta [14, 15].

Table 3.3 shows the $\hat{A}_{12}$ statistic for each scope metric (columns) and pair of generation approaches (rows). For example, the first row, the third column in Table 3.3 shows the $\hat{A}_{12}$ value for LCG vs PCG+L for the S2-Duration metric,

$\hat{A}_{12}$	S1 - Completion	S2 - Duration	S3 - Uncertainty	S4 - Killer Moves
LCG vs PCG+L	50%	23.40%	45.58%	26.70%
LCG vs PCG+R	50%	18.31%	42.74	31.10%
PCG+L vs PCG+R	50%	44.62%	47.26%	51.98%
	S5 - Permanence	S6 - Lead Change	Overall	
LCG vs PCG+L	44.91%	25.76%	26.55%	
LCG vs PCG+R	45.47%	27.43%	24.12%	
PCG+L vs PCG+R	50.03%	51.09%	47.55%	

Table 3.3: $\hat{A}_{12}$ statistic. The lower the values the better for the first approach compared. Values below 30% are highlighted in grey.

Cliff's Delta	S1 - Completion	S2 - Duration	S3 - Uncertainty	S4 - Killer Moves
LCG vs PCG+L	0 (negligible)	-0.532072 (large)	-0.088312 (negligible)	-0.465984 (medium)
LCG vs PCG+R	0 (negligible)	-0.633744 (large)	-0.14512 (negligible)	-0.378092 (medium)
PCG+L vs PCG+R	0 (negligible)	-0.10768 (negligible)	-0.054824 (negligible)	0.0396 (negligible)
	S5 - Permanence	S6 - Lead Change	Overall	
LCG vs PCG+L	-0.10184 (negligible)	-0.484864 (large)	-0.468916 (medium)	
LCG vs PCG+R	-0.090664 (negligible)	-0.451472 (medium)	-0.517696 (large)	
PCG+L vs PCG+R	0.000544 (negligible)	0.021892 (negligible)	-0.04898 (negligible)	

Table 3.4: Cliff's Delta statistic. Medium and large effect sizes are highlighted in grey.

23.40%. It indicates the probability that an observation from the first group (LCG) is greater than an observation from the second group (PCG+L). In this case, as the data represents scope differences, the lower the difference, the better. That is, in 23.40% of the runs, the differences in scope between the LCG results and the reference values in terms of the S2 metric are bigger (or worse in this case) than the corresponding differences of the PCG+L approach. Similarly, the opposite is also valid, so in 76.6% of the runs, the differences in the scope of LCG are smaller than the differences of PCG+R for the S2 metric.

Table 3.4 shows the Cliff's Delta statistic for each scope metric (columns) and pair of generation approaches (rows). It indicates the degree to which two distributions overlap. Negative values indicate that values from the first approach are smaller than those from the second. In this case, the lower the value, the better for the first approach (as the differences in the scope reference are smaller).

For both $\hat{A}_{12}$ and Cliff's Delta, we observe again that S2-Duration, S4-Killer Moves, and S6-Lead Change are the metrics that present the biggest differences between LCG and the PCG baseline. The other metrics perform similarly for the three approaches. Based on the Cliff's Delta statistic, we can conclude

that LCG can create content that is a better fit for the family’s scope than PCG+L (with a medium effect size) and PCG+R (with a large effect size).

3.5 Answer to RQs and Discussion

As an answer to RQ1, we can conclude that our LCG approach can produce results that fit the scope of the family of products in terms of the metrics studied better than a state-of-the-art approach. In particular, the differences in scope between the family of products and the content generated by our LCG approach are smaller than the corresponding differences for the baselines for all the scope metrics studied (see Table 3.1).

As an answer to RQ2, the results for our LCG approach are better than those produced by a state-of-the-art approach, generating individuals with smaller scope differences, 75.88% of the runs for the overall scope metric according to the $\hat{A}_{12}$ effect size (see Table 3.3). This can be interpreted as a medium/large difference according to the Cliff’s Delta effect size (see Table 3.4).

As an answer to RQ3, when using the seeds obtained by our Ancestor Identification operation to generate content with a state-of-the-art approach, the differences in scope are smaller than when using randomly selected seeds. However, those differences are small and only produce better scope values 52.45% of the runs.

We can explore video game content from a new perspective by mixing the two concepts of Phylogenetics from the biology domain and Product Family from the Software Product Line domain. We can leverage the already-created content with its information and knowledge to discover new content that was already there waiting to be discovered. This latent content is particularly interesting as it is ensured to follow the present design guidelines whether they are formalized or not; the content is within the family scope.

Our results demonstrate that content created with LCG is more similar to the already present content. This means the content is nearer to the production-ready state for quick incorporation into the final game. The content created via LCG also has a higher chance of following the creative intentions of the game designers.

Additionally, the structured view that the tree provides can help developers understand the product family’s relations and scope thus easing software variability. This can help developers make creative decisions on how the game

should be structured in terms of content variability. Designers can choose genetically near NPCs to place them at the same level as they are more similar and, thus, cohesive with the general intent of said level. Or if the designer wants a wide variety of NPCs for creative (narrative, aesthetics, mechanics, etc.) reasons, the phylogenetic tree provides that information.

Video games have large amounts of content and the phylogenetic tree can be a good way to introduce SPLs into video games that are already advanced in development. The fact that developers can see genetic relations among content can derive into feature models that conform to the discovered relations, similar to the approach of König et al. [27] called taxonomy mining.

Finally, referring to the simple phylogenetic tree in Figure 3.1: Would the common ancestor between the owl and the gecko be a valid product? Our results say yes, but more importantly, LCG invites developers to explore that specific content as it may produce a better individual within the video game's scope.

3.6 Threats To Validity

To address the limitations of our evaluation, we adopt the validity threat classification framework from Wohlin et al. [55], which identifies four dimensions of validity threats:

Construct Validity: This dimension examines whether the operational measures accurately reflect the true theoretical constructs they aim to represent. We addressed this threat by performing a fair comparison between our approach and the baseline using the same metrics and the same simulated player. Additionally, the six metrics used in the evaluation are accepted by the game software engineering community [11, 7].

Internal Validity: This dimension is concerned with the causality relationships established within the study. It evaluates whether the outcomes can genuinely be attributed to the treatment or intervention being researched rather than being affected by other variables. The results could be affected by implementation details; to mitigate this, we created ten different new products with each approach and executed the simulated player 50 times for each, accounting for random variation. Additionally, we have provided the implementation details and the source code.

External Validity: This dimension pertains to the generalizability of the study results to other settings, environments, or groups. The phylogenetic encoding generalizes the implementation, and thus, the distance calculation and inference are agnostic to the domain. The genetic relations present in the phylogenetic tree do not represent an ad hoc implementation for the video game content domain, as that content was genetically encoded. Our evaluation was performed over a single industry-scale video game, mitigating the lack of real problem instances. However, our findings should be replicated using other video games and alternative content to confirm the generalizability properly.

Conclusion Validity: This dimension focuses on the reliability and accuracy of the conclusions drawn from the study. It involves ensuring that the conclusions about relationships or differences are statistically and methodologically sound. We provided ten individuals for each approach comparison and executed the simulation 50 times for each individual. We also performed a statistical analysis that is widely accepted in software engineering [3] including statistical significance (Quade test and Holm’s Post Hoc) and effect size (Cliff’s Delta and $\hat{A}_{12}$) statistics.

Finally, one limitation of this approach is that it needs an existing set of products to create the phylogenetic tree. This is not the case for other PCG approaches. Our approach can render better results for specific stages of game development where there is already content created.

3.7 Related Work

This section presents the related works taking into account the terms SPL in Video Games, Content Generation in Video Games, and Phylogenetics in Software. We can see that while there are works that tackle these concepts or similar ones, none of them use phylogenetics to generate content through the lens of SPL in an industry-scale case. Our research aims to create video game content that is not only novel but also in line with the developers’ vision.

3.7.1 *SPLs in Video Games*

Some papers address the variability of video games’ content and software. Some approaches are focused on creating engines that help game developers make games more scalable, independent, and reusable. Such is the case of the Minimal Engine for Digital Games (MEnDiGa) created by Boaventura and Sarinho [9], an extension of their previous engine called FEnDiGa [47]. In MEnDiGa,

they refined the development of logic features and modules that represent and adapt game features, enabling functionality across multiple gaming platforms. Additionally, Castro and Werner [12] discuss a game prototype developed using a dynamic SPL to generate game modifications systematically. This prototype showcases the feasibility of automating the modification process, positioning the original game as the central component of the game’s functionalities.

Additional research efforts delve into developing Software Product Lines (SPLs) through re-engineering processes. Lima et al. [33, 34] introduce two studies concerning the recovery of Product Line Architecture (PLA). They implemented their previously proposed guidelines [32] to establish the PLA for the Apo-Games project [28]. Moreira et al. [41] examine empirical data from the re-engineering efforts of two open-source projects, ArgoUML and Phaser. Their findings reveal significant differences in the re-engineering processes between ArgoUML-SPL and Phaser. Common challenges are encountered in both projects, including a scarcity of tools, resulting in incomplete and inconsistent feature extractions, complexities in managing feature dependencies with a compositional approach, and the absence of a variability model to address feature constraints.

Similarly, Martinez et al. [37] share insights from their experience in creating a Software Product Line (SPL) through re-engineering system variants centered around an educational game called Robocode. They explore their findings from various angles, including the educational value, the extraction process, and the time and effort involved. Debbiche et al. [16] investigate the Apo-Games to pinpoint reusable code or artifacts. Their analysis covered five Java games, with three of these games subsequently transitioned into a composition-based SPL using *FeatureHouse* [2]. They claim that maintaining the testability of the SPL ensures accurate code transformations and recommend the incremental incorporation of new features to ease the extraction process.

3.7.2 *Procedural Content Generation from Game Software Engineering*

Relevant to PCG, Preuss et al. [45] examine the interplay between quality and diversity in PCG for game development. Their research involved an experimental evaluation of various algorithms and distance measures using a tool designed for generating game levels. They concluded that the Niching Evolutionary Algorithm 2 (NEA2) effectively balances quality and diversity, contingent upon the employment of a robust distance function. Subsequently,

Gravina et al. [20] characterized quality diversity as a pivotal search strategy within search-based PCG.

The study conducted by Melotti et al. [40] introduces and implements the Deluged Novelty Search Local Competition algorithm (D-NSLC), which utilizes morphological niches to promote solution diversity in PCG for a game. D-NSLC segments the population into distinct niches and targets the optimal individuals within each while exploring the search space. They conducted an experiment within a roguelike video game context using four different setups. The findings underscored the advantages of the Novelty Search, notably its significant contribution to generating diverse individuals.

Finally, Blasco et al. explored the application of Search-Based Software Engineering (SBSE) for content generation [7, 8]. These studies utilized an evolutionary algorithm steered by simulations that incorporate the generated content. They use an evolutionary algorithm known as Evolutionary Model Generation (EMoGen) to produce software models quickly and efficiently. The research demonstrated that models generated by EMoGen for the commercial video game Kromaia were comparable in quality to those created manually by developers but took significantly less time.

None of the aforementioned studies leverage the idea of latent content with genetic divergence points and ancestry branches. This work opens a new path where the focus is that the content is aligned with the scope precisely with the bottom-up relationship. In the video game domain, the scope is crucial because the developers' vision for the game is critical and difficult to formalize.

3.7.3 Phylogenetics in Software

To the best of our knowledge, there is no study that applies phylogenetics to software. However, the study by König et al. [27] focuses on enhancing the understanding and application of software variability through the technique of taxonomy mining. The authors present a methodological approach aimed at extracting and utilizing taxonomies to manage variability in software product lines better. Their approach is applied to generate taxonomy graphs of different SPLs in Software Engineering. Taxonomy and Phylogeny are closely related sciences in the realm of biology. Taxonomy focuses on the study of the classification of species, and Phylogeny focuses on the study of evolutionary relationships between organisms. They have different objectives using the same information.

3.8 Conclusion

Our LCG approach proves itself to be a valid approach for content creation in video games. The evaluation has demonstrated its capabilities to identify gaps within a product family, hidden content waiting to be discovered. The content created via LCG is more similar to the current scope of an industry-scale video game.

Phylogenetics can manage variability by pointing out genetic relations among individuals of the same product family. Identifying these relations can be key for the future of software variability, providing developers with a full view of an SPL at a glance. This can enhance strategic decision-making regarding the development and deployment of additional products.

Specifically, in video games, the phylogenetic tree can help designers arrange and manage their content while guiding the next steps in the creation of new content. Also, our approach can potentially be used in other domains beyond video game content creation. Genetic divergence points and ancestry branches can provide new perspectives to variability thanks to the phylogenetic approach.

Acknowledgements

This work has been partially supported by the following: Spanish National R+D+i Plan and ERDF funds under the Projects VARNETICA (CNS2023-145422) and VARIATIVA (PID2021-128695OB-I00); Centro para el Desarrollo Tecnológico y la Innovación (CDTI) (SAV-20221011); Gobierno de Aragón (Spain) (Research Group S05 20D); And by FEDER/Ministry of Science, Innovation and Universities/Junta de Andalucía/State Research Agency/CDTI with the following grants: TASOVA PLUS research network (RED2022-134337-T).

Bibliography

- [1] Apostolos Ampatzoglou and Ioannis Stamelos. “Software engineering research for computer games: A systematic review”. In: *Information and Software Technology* 52.9 (2010), pp. 888–901. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2010.05.004> (cit. on p. 111).
- [2] Sven Apel, Christian Kästner, and Christian Lengauer. “Language-Independent and Automated Software Composition: The FeatureHouse Experience”. In: *IEEE Transactions on Software Engineering* 39.1 (2013), pp. 63–79. DOI: 10.1109/TSE.2011.120 (cit. on p. 134).
- [3] Andrea Arcuri and Lionel Briand. “A Hitchhiker’s Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering”. In: *Softw. Test. Verif. Reliab.* 24.3 (May 2014), pp. 219–250. ISSN: 0960-0833. DOI: 10.1002/stvr.1486 (cit. on pp. 122, 128, 133).
- [4] Earl T. Barr et al. “The plastic surgery hypothesis”. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, 306–317. ISBN: 9781450330565. DOI: 10.1145/2635868.2635898 (cit. on p. 111).
- [5] D.A. Baum and S.D. Smith. *Tree Thinking: An Introduction to Phylogenetic Biology*. Macmillan Learning, 2012. ISBN: 9781936221165 (cit. on p. 114).
- [6] T. Biggerstaff and C. Richter. “Reusability Framework, Assessment, and Directions”. In: *IEEE Software* 4.2 (1987), pp. 41–49. DOI: 10.1109/MS.1987.230095 (cit. on p. 111).
- [7] Daniel Blasco et al. “An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering”. In: *Journal of Systems and Software* 171 (2021), p. 110804 (cit. on pp. 115, 120, 122, 132, 135).
- [8] Daniel Blasco et al. “Procedural content improvement of game bosses with an evolutionary algorithm”. In: *Multimedia Tools and Applications* 82.7 (2023), pp. 10277–10309 (cit. on p. 135).
- [9] Filipe M. B. Boaventura and Victor Travassos Sarinho. “MEnDiGa: A Minimal Engine for Digital Games”. In: *Int. J. Comput. Games Technol.* 2017 (2017), 9626710:1–9626710:13. DOI: 10.1155/2017/9626710 (cit. on p. 133).

- [10] Andrew VZ Brower and Randall T Schuh. *Biological Systematics: Principles and Applications*. Cornell University Press, 2021 (cit. on p. 117).
- [11] Cameron Browne and Frederic Maire. “Evolutionary game design”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 2.1 (2010), pp. 1–16 (cit. on pp. 121, 122, 132).
- [12] Diego Castro and Cláudia Werner. “Rebuilding games at runtime”. In: IEEE, 2021, pp. 73–77. DOI: 10.1109/ASEW52652.2021.00025 (cit. on p. 134).
- [13] Paul Clements and Linda Northrop. *Software product lines*. Addison-Wesley Boston, 2002 (cit. on p. 111).
- [14] Norman Cliff. “Dominance statistics: Ordinal analyses to answer ordinal questions.” In: *Psychological bulletin* 114.3 (1993), p. 494 (cit. on pp. 120, 129).
- [15] Norman Cliff. *Ordinal methods for behavioral data analysis*. Psychology Press, 2014 (cit. on pp. 120, 129).
- [16] Jamel Debbiche et al. “Migrating Java-based Apo-Games into a composition-based software product line”. In: ACM, 2019, 18:1–18:5. DOI: 10.1145/3336294.3342361 (cit. on p. 134).
- [17] Mark Gabel and Zhendong Su. “A study of the uniqueness of source code”. In: *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE ’10. New York, NY, USA: Association for Computing Machinery, 2010, 147–156. ISBN: 9781605587912. DOI: 10.1145/1882291.1882315 (cit. on p. 111).
- [18] Roberto Gallotta, Kai Arulkumaran, and L. B. Soros. “Evolving spaceships with a hybrid L-system constrained optimisation evolutionary algorithm”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO ’22. New York, NY, USA: Association for Computing Machinery, 2022, 711–714. ISBN: 9781450392686. DOI: 10.1145/3520304.3528775 (cit. on p. 120).
- [19] Salvador García et al. “Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power”. In: *Information Sciences* 180.10 (2010), pp. 2044–2064 (cit. on pp. 120, 128, 129).

- [20] Daniele Gravina et al. “Procedural content generation through quality diversity”. In: *2019 IEEE Conference on Games (CoG)*. IEEE. 2019, pp. 1–8 (cit. on p. 135).
- [21] R. J. Grissom and J. J. Kim. *Effect sizes for research: A broad practical approach*. Mahwah, NJ: Earlbaum, 2005 (cit. on p. 129).
- [22] Mark Hendrikx et al. “Procedural content generation for games: A survey”. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 9.1 (2013). ISSN: 1551-6857. DOI: 10.1145/2422956.2422957 (cit. on p. 111).
- [23] Kent Holsinger and Bruce Weir. “Genetics in geographically structured populations: Defining, estimating and interpreting FST”. In: *Nature reviews. Genetics* 10 (Oct. 2009), pp. 639–50. DOI: 10.1038/nrg2611 (cit. on p. 114).
- [24] Robin Hunicke, Marc LeBlanc, Robert Zubek, et al. “MDA: A formal approach to game design and game research”. In: *Proceedings of the AAAI Workshop on Challenges in Game AI*. Vol. 4. 1. San Jose, CA. 2004, p. 1722 (cit. on p. 122).
- [25] T. Capers Jones. “Reusability in Programming: A Survey of the State of the Art”. In: *IEEE Transactions on Software Engineering* SE-10.5 (1984), pp. 488–494. DOI: 10.1109/TSE.1984.5010271 (cit. on p. 111).
- [26] Bohyun Kim. “Game mechanics, dynamics, and aesthetics”. In: *Library technology reports* 51.2 (2015), pp. 17–19 (cit. on p. 119).
- [27] Christoph König et al. “True Variability Shining Through Taxonomy Mining”. In: *Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume A*. SPLC ’23. New York, NY, USA: Association for Computing Machinery, 2023, 182–193. ISBN: 9798400700910. DOI: 10.1145/3579027.3608989 (cit. on pp. 132, 135).
- [28] Jacob Krüger et al. “Apo-games: a case study for reverse engineering variability from cloned Java variants”. In: ed. by Thorsten Berger et al. ACM, 2018, pp. 251–256. DOI: 10.1145/3233027.3236403 (cit. on p. 134).
- [29] Robert G. Lanergan and Charles A. Grasso. “Software Engineering with Reusable Designs and Code”. In: *IEEE Transactions on Software Engineering* SE-10.5 (1984), pp. 498–501. DOI: 10.1109/TSE.1984.5010273 (cit. on p. 111).

- [30] M. Lenz, H.A. Schmid, and P.F. Wolf. “Software Reuse through Building Blocks”. In: *IEEE Software* 4.4 (1987), pp. 34–42. DOI: 10.1109/MS.1987.231062 (cit. on p. 111).
- [31] Vladimir I Levenshtein. “Binary codes capable of correcting spurious insertions and deletions of ones”. In: *Problems of Information Transmissions* 1.1 (1965), pp. 8–17 (cit. on p. 115).
- [32] Crescencio Lima, Christina Chavez, and Eduardo Santana de Almeida. “Investigating the Recovery of Product Line Architectures: An Approach Proposal”. In: ed. by Goetz Botterweck and Claudia Werner. Cham: Springer International Publishing, 2017, pp. 201–207. ISBN: 978-3-319-56856-0 (cit. on p. 134).
- [33] Crescencio Lima et al. “Recovering the product line architecture of the apo-games”. In: ed. by Thorsten Berger et al. ACM, 2018, pp. 289–293. DOI: 10.1145/3233027.3236398 (cit. on p. 134).
- [34] Crescencio Lima et al. “Towards an Automated Product Line Architecture Recovery: The Apo-Games Case Study”. In: ACM, 2018, pp. 33–42. DOI: 10.1145/3267183.3267187 (cit. on p. 134).
- [35] Jialin Liu et al. “Deep learning for procedural content generation”. In: *Neural Computing and Applications* 33.1 (2021), pp. 19–37 (cit. on p. 111).
- [36] Niko Mäkitalo et al. “On opportunistic software reuse”. In: *Computing* 102 (2020), pp. 2385–2408 (cit. on p. 111).
- [37] Jabier Martinez, Xhevahire Tërnavaj, and Tewfik Ziadi. “Software Product Line Extraction from Variability-Rich Systems: The Robocode Case Study”. In: SPLC ’18. Association for Computing Machinery, 2018, 132–142. ISBN: 9781450364645. DOI: 10.1145/3233027.3233038 (cit. on p. 134).
- [38] M. D. McIlroy. “Mass-produced software components”. In: *Proc. NATO Conf. on Software Engineering, Garmisch, Germany* (1968) (cit. on p. 111).
- [39] Michael McShaffry. *Game Coding Complete*. Paraglyph Publishing, 2003. ISBN: 1932111751 (cit. on p. 111).
- [40] Alexandre Santos Melotti and Carlos Henrique Valerio de Moraes. “Evolving roguelike dungeons with deluged novelty search local competition”. In: *IEEE Transactions on Games* 11.2 (2018), pp. 173–182 (cit. on p. 135).

-
- [41] Rodrigo André Ferreira Moreira et al. “Open-source software product line extraction processes: the ArgoUML-SPL and Phaser cases”. In: *Empirical Software Engineering* 27.4 (2022) (cit. on p. 134).
- [42] P. Naur, B. Randell, and NATO Science Committee. *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October, 1968*. Scientific Affairs Division, NATO, 1969 (cit. on p. 111).
- [43] Masatoshi Nei. *Molecular Evolutionary Genetics*. Columbia University Press, 1987. ISBN: 0231063202 (cit. on p. 113).
- [44] Cristiano Politowski et al. “Are game engines software frameworks? a three-perspective study”. In: *Journal of Systems and Software* 171 (2021), p. 110846 (cit. on p. 111).
- [45] Mike Preuss, Antonios Liapis, and Julian Togelius. “Searching for good and diverse game levels”. In: *2014 IEEE Conference on Computational Intelligence and Games*. IEEE. 2014, pp. 1–8 (cit. on p. 134).
- [46] Naruya Saitou and Masatoshi Nei. “The neighbor-joining method: a new method for Reconstructing Phylogenetic Trees”. In: *Mol Biol Evol* 4.4 (1987), pp. 406–425 (cit. on p. 116).
- [47] Victor T. Sarinho, Antônio L. Apolinário, and Eduardo S. Almeida. “A Feature-Based Environment for Digital Games”. In: ed. by Marc Herrlich, Rainer Malaka, and Maic Masuch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 518–523. ISBN: 978-3-642-33542-6 (cit. on p. 133).
- [48] Jesse Schell. *The Art of Game Design: A book of lenses*. CRC press, 2008 (cit. on p. 122).
- [49] Adam Summerville et al. “Procedural content generation via machine learning (PCGML)”. In: *IEEE Transactions on Games* 10.3 (2018), pp. 257–270 (cit. on p. 111).
- [50] Julian Togelius et al. “Search-Based Procedural Content Generation: A Taxonomy and Survey”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 172–186. DOI: 10.1109/TCIAIG.2011.2148116 (cit. on p. 111).
- [51] András Vargha and Harold D. Delaney. “A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong”. In: *Journal of Educational and Behavioral Statistics* 25.2 (2000), pp. 101–

132. eprint: <http://jeb.sagepub.com/content/25/2/101.full.pdf+html> (cit. on pp. 120, 129).
- [52] G Weyenberg and R Yoshida. *Chapter 12-Reconstructing the Phylogeny: Computational Methods. Algebraic and Discrete Mathematical Methods for Modern Biology*. RS Robeva. 2015 (cit. on p. 115).
- [53] Grady Weyenberg and Ruriko Yoshida. “Chapter 12 - Reconstructing the Phylogeny: Computational Methods”. In: *Algebraic and Discrete Mathematical Methods for Modern Biology*. Ed. by Raina S. Robeva. Boston: Academic Press, 2015, pp. 293–319. ISBN: 978-0-12-801213-0. DOI: <https://doi.org/10.1016/B978-0-12-801213-0.00012-5> (cit. on p. 114).
- [54] Tom Wijman. *Global Games Market Report*. [Online; accessed 17-November-2023]. 2023. URL: <https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2023-free-version> (cit. on p. 111).
- [55] Claes Wohlin et al. *Experimentation in software engineering*. Springer Science & Business Media, 2012 (cit. on p. 132).
- [56] Yuzhong Zhang, Guixuan Zhang, and Xinyuan Huang. “A Survey of Procedural Content Generation for Games”. In: *2022 International Conference on Culture-Oriented Science and Technology (CoST)*. 2022, pp. 186–190. DOI: 10.1109/CoST57098.2022.00046 (cit. on p. 111).
- [57] Meng Zhu and Alf Inge Wang. “Model-driven Game Development: A Literature Review”. In: *ACM Comput. Surv.* 52.6 (2020), 123:1–123:32. DOI: 10.1145/3365000 (cit. on p. 115).

Part III

Discussion

Discussion

Content creation and variability management arise as key factors in video game development. Through GSE research, we manage to tackle them by using SE techniques.

Throughout the pages of this thesis, we have portrayed different GSE approaches that tackle the various challenges of video game software creation. More specifically, the bottleneck is content creation for video games. We have studied the overall state of GSE research and the current approaches toward content creation. Our findings revealed that GSE techniques positively impact video game creation, more so in a Model-Driven fashion, as MDD is used industrially for video games in game engines such as Unreal Engine and Unity which have visual coding tools. The most notorious case of software modeling in video games is the rendering code. It is mainly created through model-based tools to make the shaders for the GPU. These tools are present in almost all the engines and programs used for video game creation: Unity, Unreal Engine, Blender, Adobe Substance 3D Designer, Autodesk Maya, and others.

Our work focuses on using software models as the central artifact when creating content. Whether by comparing Software Product Lines (SPLs) with Clone and Own (CaO) to create bosses [1], by creating a Mixed-Initiative Genetic Algorithm to generate creative content [3], or by developing a phylogenetic approach to identify and generate latent content within a family of products [2], MDD is present in all aspects of this work. Models have been demonstrated to be the cornerstone of video game content specification. They open the content creation doors to non-engineers thanks to their abstraction level. They streamline some processes and form new paths for creative work. An important note is that direct written code is not disregarded by developers in the least. However, everyday software models play a bigger role in the video game development process.

Additionally, every work is aimed at the video game industry. For this reason, we have involved our industrial partners in the research process. To ensure the industrial relevance of the GSE approaches, they are evaluated with the help of industrial case studies or expert practitioners. Also, the use of multipurpose and widely used content for video games helps generalize our findings.

Research Questions

With the work present in this thesis, we aim to improve the bottleneck of content by answering the following research questions.

RQ1. Can SE techniques be used to improve **content generation** in video games? If so, how do these techniques need to be adapted to tackle the video game domain's particularities?

Content generation can come in a myriad of ways, as “content” can be anything present in a video game. We have focused on multipurpose content that considers different roles in the development process. We make use of the final bosses in the game Kromaia in the papers **JSS’23** [1] and **SPLC’24** [2]. These game elements specify work made by designers (balance, fun), programmers (behavior, physics), and artists (visual representation). Another content taken into account is particle systems (PSs) in the paper **IST’24** [3]. They represent a mix of technical work and artistic taste; typically, the role tasked to make these PSs are called technical artists. For this question, all three studies take effect.

Our first research published, **JSS’23** [1], compares SPL with CaO inside two different development paradigms: Model-Driven Development (MDD) and Code-Driven Development (CDD). The experiment results show that SPLs generally outperform CaO in GSE, and the differences are more pronounced under the CDD paradigm. These are the three primary metrics taken into account:

- *Correctness*: Bosses developed using SPLs had better correctness (over 23%) than those developed with CaO in both MDD and CDD paradigms. Both experienced and inexperienced subjects developed more correct bosses when using SPL instead of CaO.
- *Efficiency*: In CDD, SPL provided significant improvements in efficiency (51%). However, there were no significant differences in MDD. As expected, experienced developers were more efficient than inexperienced ones, regardless of the approach or paradigm used.
- *Satisfaction*: In CDD, SPL provided significant improvements in satisfaction (13%). The effect under the MDD paradigm was small for Perceived Ease of Use (PEOU) in favor of SPL and very small or negligible for Perceived Usefulness (PU) and Intention to Use (ITU) in favor of CaO. The experienced subjects were more generous in rating their satisfaction.

SPLs may play a different role for GSE than in traditional SE. They can be valuable for balancing or generating new content. Also, developers feared losing control over their software when using SPLs.

Our second research, published, **IST’24** [3], proposes Mixed-Initiative, adapting the MAP-Elites algorithm [5], incorporating the human in the process to guide the fitness function. We leverage the quality diversity that MAP-Elites provides to present different solutions to developers. This way, regardless of the role, the developer chooses the best models (based on their creative preference)

that update the fitness function. The process is iterative until the developer is satisfied with one of the diverse results. The approach is evaluated using Particle Systems (PS), a popular type of content used to create visual effects in video games.

The evaluation involved five professional game developers who used the tool to create PS for their games. They were asked to create five different types of PS: Fire, Rain, Snow, Smoke, and Sparks. A visual effects expert also created handmade PS for each type, which were used as a baseline for comparison. Quantitatively, in four out of five categories (Fire, Rain, Smoke, and Snow), the game developers preferred the co-created PS over the handmade ones. Also, the co-created PS were generated by non-experts in 5 minutes, compared to 15 minutes taken by the expert to create the handmade versions. On the qualitative side, the Mixed-Initiative approach participants agreed on the usefulness of the approach, emphasizing the ease of use and the benefits of real-time visualization. Also, developers created each PS with a specific game in mind. They were not looking for a universally "best" PS but one that fits the context of their particular game.

The approach is more accessible and efficient than learning to use custom tools, and developers can tweak the generated PS to their needs. The approach does not require a high level of PS knowledge to be used effectively.

Our third research published, **SPLC'24** [2], introduces an approach that creates content by applying biological insights to map the landscape of a software product family. This involves creating a "phylogenetic tree" that visualizes the relationships and divergence between different software products. In this case, the family consists of all the Kromaia's Non-Playable Characters (NPCs), which are all the enemies in the game, including bosses. This approach is used to identify "latent" products that don't exist in the original product family. The identified ancestors can be reconstructed from existing products (descendants) to create new content. The technique is called Latent Content Generation (LCG).

To evaluate the content, we use established video game metrics (Completion, Duration, Uncertainty, Killer Moves, Permanence, and Lead Change) to assess how well the generated content fits within the scope of the game. These metrics are retrieved via a simulated player that faces the NPCs in combat. To compare these metrics, we use the state-of-the-art Procedural Content Generation (PCG) approach by Gallota et al. [4] as a baseline. The main finding is that content generated using the LCG approach aligns more closely with the original scope of the Kromaia product family compared to content generated by

the PCG approach. The LCG approach produced content with smaller scope differences 75.88% of the time. Thus, the generated content is more likely to adhere to the design guidelines and creative vision of the developers.

Finally, answering **RQ1**: We can effectively use SE techniques to improve content generation. We should not use them as is, but they need adaptation to the particularities of the video game domain. When developing a new approach for generating content, we need to take into account that video game developers often lack engineering knowledge and have a higher focus on other aspects. Game developers view their work as an artisanal craft rather than an industrial process. They want to be in charge of all the steps in the development. There has to be a balance in easing the content creation without taking away the developers' agency over their craft. Another key aspect when providing content generation approaches is to be aware that every game, in the end, is a subjective product. A video game needs, for example, to be fun, to feel good, or to seem cool. This subjectivity puts humans, again, at the center of **content generation**.

RQ2. Can SE techniques be used to improve **variability management** in video games? If so, how do these techniques need to be adapted to tackle the video game domain's particularities?

Once the content is created, it needs to be properly managed. Video game developments can span multiple years of iteration. So, it is key to have a clear understanding of all the game content that is created, discarded, and modified during that time. For this question, the first and last studies take effect.

Our first research published, **JSS'23** [1], presents two different SPLs for final bosses in Kromaia. One compositional SPL for the MDD paradigm and one annotative SPL for the CDD paradigm. Subjects reported that these SPLs are easier, more accessible, less error-prone, and clearer to understand than manually managing all the content. These are key features for non-engineers to understand all the content better.

These SPLs manage variability through the use of variation points, which are linked to a predefined set of features determined by domain experts, allowing for systematic reuse and configuration. In contrast, CaO involves duplicating and modifying existing code bases, offering flexibility but often at the cost of maintainability and consistency because of the introduction of technical debt into the projects.

Our last research, published as **SPLC'24** [2], encodes reusable software elements, such as enemies from Kromaia, as forms of genetic information. The

method constructs a phylogenetic tree that reveals ancestry and divergence among products. This tree helps identify genetic relationships among individuals. This provides a structured view of the product family and offers a new perspective on variability. The developer can point out at a glance the individuals who are susceptible to collapse into a single one by refactoring the software. This LCG approach is based on creating content through variability management by identifying the gaps in the phylogenetic tree.

Finally, answering **RQ2**: When managing variability, we must take into account particularities of video game development teams and content. These teams are big and heterogeneous. Many roles are involved in the creation of video games, and most of them are non-technical roles. And content is also often non-technical, and it is affected by subjective requirements. So when applying software management techniques, we have to take into account that creative people are going to build creative content. With SPLs, we can manage code variability, and with phylogenetics, we can manage creative variability. By encoding the different individuals, we can provide a structured view of the video game elements. This subjectivity puts humans, again, at the center of **variability management**.

Future Work

While this thesis has contributed to the understanding of software development practices and content generation in Game Software Engineering, several promising avenues for future research remain open.

One key direction involves extending the empirical work presented here to additional video game projects currently in development. Studying games at different stages of production could provide further insights into how SPLs can be applied throughout the development lifecycle, particularly in managing variability and reuse for multi-platform releases across computers, consoles, and mobile devices. Such investigations would help generalize findings and assess the adaptability of SPL approaches in a range of development contexts.

Another area for exploration is the application of the co-creation approach to creative domains beyond Particle Systems. While the current work demonstrates its potential in that specific context, evaluating its effectiveness in other types of content, such as levels, character designs, or environmental assets, could provide a broader understanding of its utility. In addition, the development of new fitness functions and genetic operators tailored to these domains could enhance the ability of human creators to guide the search process and

navigate the creative space more intuitively. Future studies may also focus on iteratively evolving and refining existing content, further supporting creativity within game development workflows.

The LCG approach introduced in this research is currently being extended for submission to a peer-reviewed journal. This ongoing work aims to refine the method and explore its application to not only generate latent content but also help in the identification and isolation of features among content. This extension explores the approach as a way of extracting features with the objective of formalizing variability. The phylogenetic approach opens new possibilities for understanding variability in GSE content as it can earmark the hotspots of poor variability management, portraying the genetic distance among individuals.

Furthermore, the rise of large language models presents an exciting opportunity for GSE. These models could support the creative process by generating descriptive metadata, proposing meaningful content variations, or assisting in design decisions through natural language interaction. This integration has the potential to significantly enhance both co-creation and LCG techniques.

Together, these research directions offer the opportunity to deepen our understanding of content production in video games and expand the relevance of the techniques developed in this thesis to a broader set of software and creative domains.

Part IV

Conclusions

Conclusions

Video games are a fertile domain in which to improve software techniques. Software techniques are the key to improving video games.

This thesis has explored the application of advanced software engineering techniques to address two of the most pressing challenges in Game Software Engineering (GSE): managing variability and accelerating content creation. Through empirical studies, we have shown how structured approaches such as Software Product Lines (SPLs), mixed-initiative co-creation, and bio-inspired phylogenetic analysis can offer concrete improvements in the way game content is designed, reused, and created.

The key insight emerging from this work is the central and irreplaceable role of the human element in game development. Unlike traditional software systems, video games are creative, emotionally charged products that must resonate with players through visuals, mechanics, narrative, and interactivity. Even as automated methods and generative techniques become more sophisticated, their success depends heavily on human guidance, intuition, and aesthetic judgment. Whether in co-creative systems that combine evolutionary algorithms with software models or in phylogenetic reconstructions of latent content, it is the developer's input that steers the process toward relevance and quality. The consistent improvements in the metrics seen across our empirical evaluations reinforce this point: software techniques in GSE must support, not replace, human creativity.

Finally, video games represent a fertile domain for innovating and refining software engineering practices. Their complexity, variability, and constant demand for new content push the limits of existing methods, encouraging experimentation and interdisciplinary approaches. At the same time, advances in software engineering are the key to improving how games are built, scaled, and maintained. By addressing the challenges unique to GSE, these techniques can ultimately lead to better tools and more efficient processes that can span beyond the video game industry.

