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

Jorge Chueca[a,b], Carlos Cetina[a,c], Oscar Pastor[b], Jaime Font[a]

[a] *SVIT Research Group, Universidad San Jorge, Zaragoza, Spain*
[b] *PROS Research Centre, Universitat Politècnica de València, Valencia, Spain*
[c] *University College London, London, United Kingdom*

Abstract

Context: 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.

Objective: 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.

Method: 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.

Results: 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.

Conclusion: 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.

Keywords: Game Software Engineering, Search-Based Software Engineering, Model-Driven Engineering, Co-creation

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 [1]. 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 [2] and Unreal Engine [3]. They allow video game developers to create content directly using code (e.g., C++ or C#) or software models [4].

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) [5] paradigm. The most notable example is node-based scripting to create shaders: Unreal Engine has Material Editor [6], and Unity Engine has Shader Graph [7].

Nowadays, the generation of content is the bottleneck of the industry. Super productions like Cyberpunk 2077 can take almost a decade to make [8]. 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. [10]. 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) [11] with the algorithm Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) [12] 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 [13]. 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 presents the motivation and background of the paper. Section 3 outlines works related to this study. Section 4 presents the proposed co-creation approach. Section 5 shows the evaluation performed. Section 6 shows the discussion of the results, and, finally, Section 8 concludes the paper.

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 [14]. 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) [15] 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* [16]. 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 [17].

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 [18], 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.

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.

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 [19]. 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 [12]. 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* [20]. 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 [21].

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. [22]. 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. [23]. 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 [24] 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.

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. [25] 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 [26] 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. [10], 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.

3.3. Humans in the model-generation algorithm

There are many contributions that delve into human intervention in model-generation algorithms. Kessentini et al.[27] propose an interactive multi-objective approach to address the challenges arising from the frequent evolution of metamodels, 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 [28] 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.[29] 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 solutions like heuristics, can offer valuable insights because of the abstract nature of models.

Bavota et al. [30] 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. [31] 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. [32] create smart agents with Reinforcement Learning to ease the testing tasks of video games.

4. Co-creation Approach

Fig. 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.

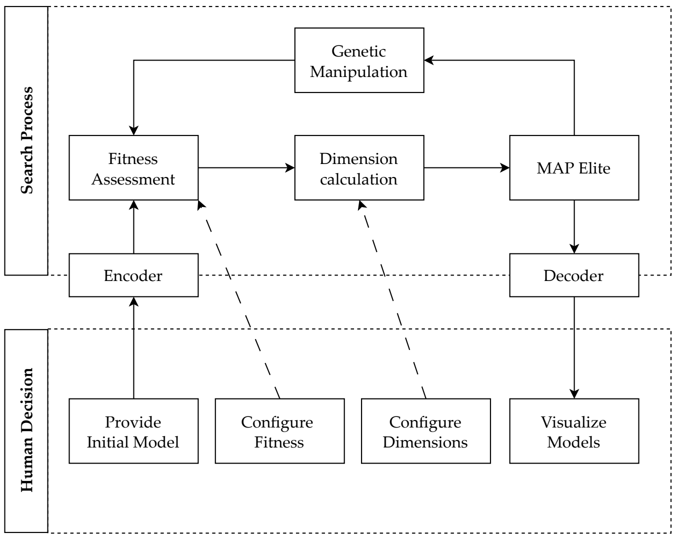

Figure 1: Co-Creation approach overview.

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 [12] 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 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 1 Mixed-Initiative MAP-Elites.

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

Algorithm 2 Evolve individuals and store in MAP Elite

```
 1: function evolve_individuals($n, \mathcal{M}, \mathcal{F}$)   ▷ number of individuals to evolve (n), current MAP elite ($\mathcal{M}$) and fitness models ($\mathcal{F}$)
 2:     for $1 \rightarrow n$ do
 3:         $r \leftarrow$ random_selection($\mathcal{M}$)        ▷ randomly select an individual r from $\mathcal{M}$
 4:         $x \leftarrow$ genetic_manipulation($r$)              ▷ create a copy of r and randomly modify a gene
 5:         $x_f \leftarrow$ fitness_assessment($x, \mathcal{F}$) ▷ calculate the fitness of x and store it in $x_f$
 6:         $x_d \leftarrow$ dimension_calculation($x$)           ▷ 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   ▷ if the cell is empty or its occupant's fitness is $\leq x_f$, then
 8:             $\mathcal{M}(x_d) \leftarrow x$                   ▷ store the new individual x in the appropriate cell of the MAP $\mathcal{M}$
 9:         end if
10:     end for
11:     return $\mathcal{M}$                                      ▷ returns the map withe the new individuals
12: end function
```

4.1.1. 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 [33]. 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.

4.1.2. 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 oth-

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:              $d_f \leftarrow d_f +$ calculate_distance($g_f$,$g_x$)    ▷ accumulate distances of each pair of genes in $d_f$
7:          end for
8:          $d_{all} \leftarrow d_{all} + (d_f / genes.size)$    ▷ 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
```

erwise. 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.

4.1.3. 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.

4.1.4. 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).

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.

4.2.1. 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.

4.2.2. 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.

4.2.3. 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 requires that be 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.

4.2.4. 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, there by encouraging the diversity of the results and facilitating the creative process.

4.2.5. 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.

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?

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.

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.

5.2.1. Particle System Modeling Language (PSML)

The Particle System Modeling Language (PSML) is a DSL that allows the creation of models representing PS. Fig. 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.

5.2.2. 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:

$$C_{max} = max(R, G, B)$$

$$C_{min} = min(R, G, B)$$

$$\Delta = C_{max} - C_{min}$$

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

- **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 *MinMaxCurve* data type that can represent a constant *float*, a random value between two constants, or

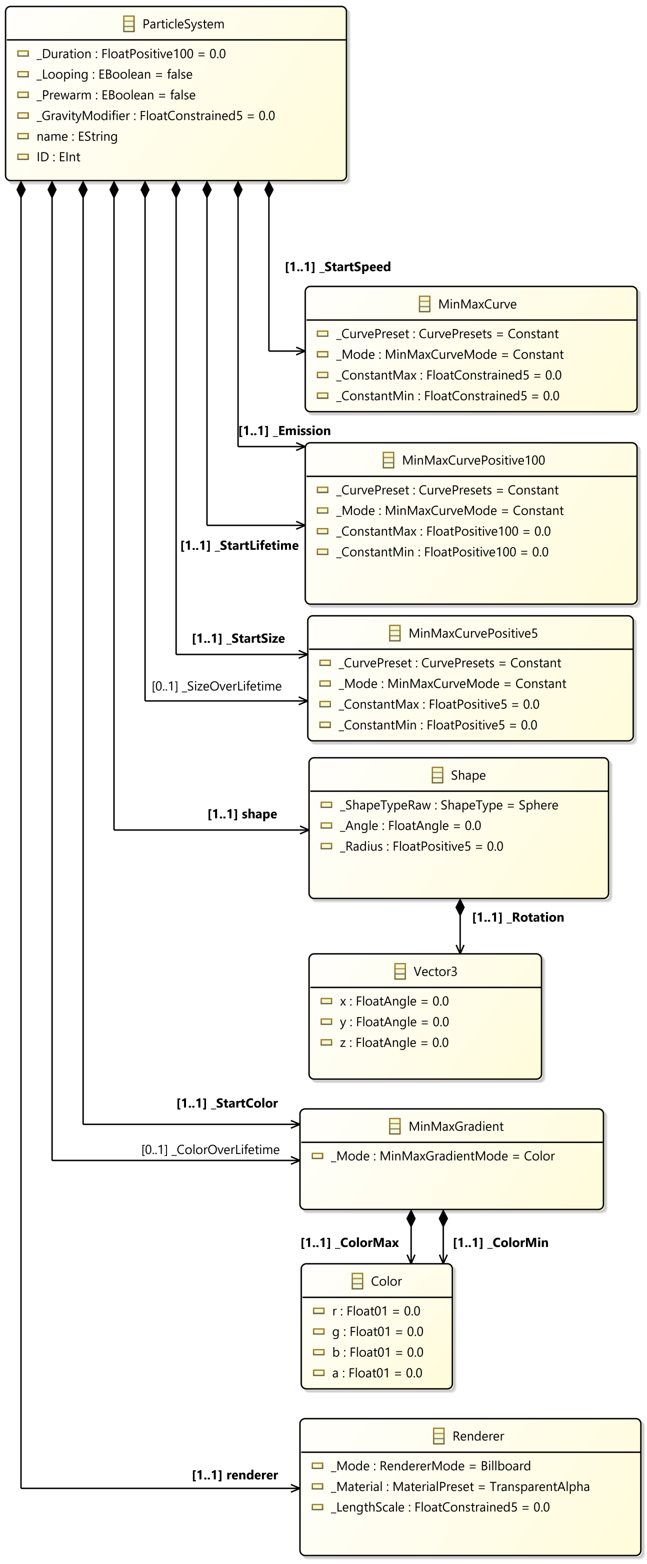
ParticleSystem
_Duration : FloatPositive100 = 0.0
_Looping : EBoolean = false
_Prewarm : EBoolean = false
_GravityModifier : FloatConstrained5 = 0.0
name : EString
ID : EInt
[1..1] _StartSpeed
MinMaxCurve
_CurvePreset : CurvePresets = Constant
_Mode : MinMaxCurveMode = Constant
_ConstantMax : FloatConstrained5 = 0.0
_ConstantMin : FloatConstrained5 = 0.0
[1..1] _Emission
MinMaxCurvePositive100
_CurvePreset : CurvePresets = Constant
_Mode : MinMaxCurveMode = Constant
_ConstantMax : FloatPositive100 = 0.0
_ConstantMin : FloatPositive100 = 0.0
[1..1] _StartLifetime
[1..1] _StartSize
MinMaxCurvePositive5
_CurvePreset : CurvePresets = Constant
_Mode : MinMaxCurveMode = Constant
_ConstantMax : FloatPositive5 = 0.0
_ConstantMin : FloatPositive5 = 0.0
[0..1] _SizeOverLifetime
Shape
_ShapeTypeRaw : ShapeType = Sphere
_Angle : FloatAngle = 0.0
_Radius : FloatPositive5 = 0.0
[1..1] shape
[1..1] _Rotation
Vector3
x : FloatAngle = 0.0
y : FloatAngle = 0.0
z : FloatAngle = 0.0
[1..1] _StartColor
MinMaxGradient
_Mode : MinMaxGradientMode = Color
[0..1] _ColorOverLifetime
[1..1] _ColorMax
[1..1] _ColorMin
Color
r : Float01 = 0.0
g : Float01 = 0.0
b : Float01 = 0.0
a : Float01 = 0.0
Renderer
_Mode : RendererMode = Billboard
_Material : MaterialPreset = TransparentAlpha
_LengthScale : FloatConstrained5 = 0.0
[1..1] renderer

Figure 2: Excerpt of the Particle System Modeling Language metamodel.

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.

5.2.3. *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 [34], and based on the pseudo-code provided for the MAP-Elites algorithm [12]. 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 [2], 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.

Fig. 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, Fig. 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.

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 1 shows the answers. The Subject column is the subject represented by a letter. The Age column is the age in years. The Dev. 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 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

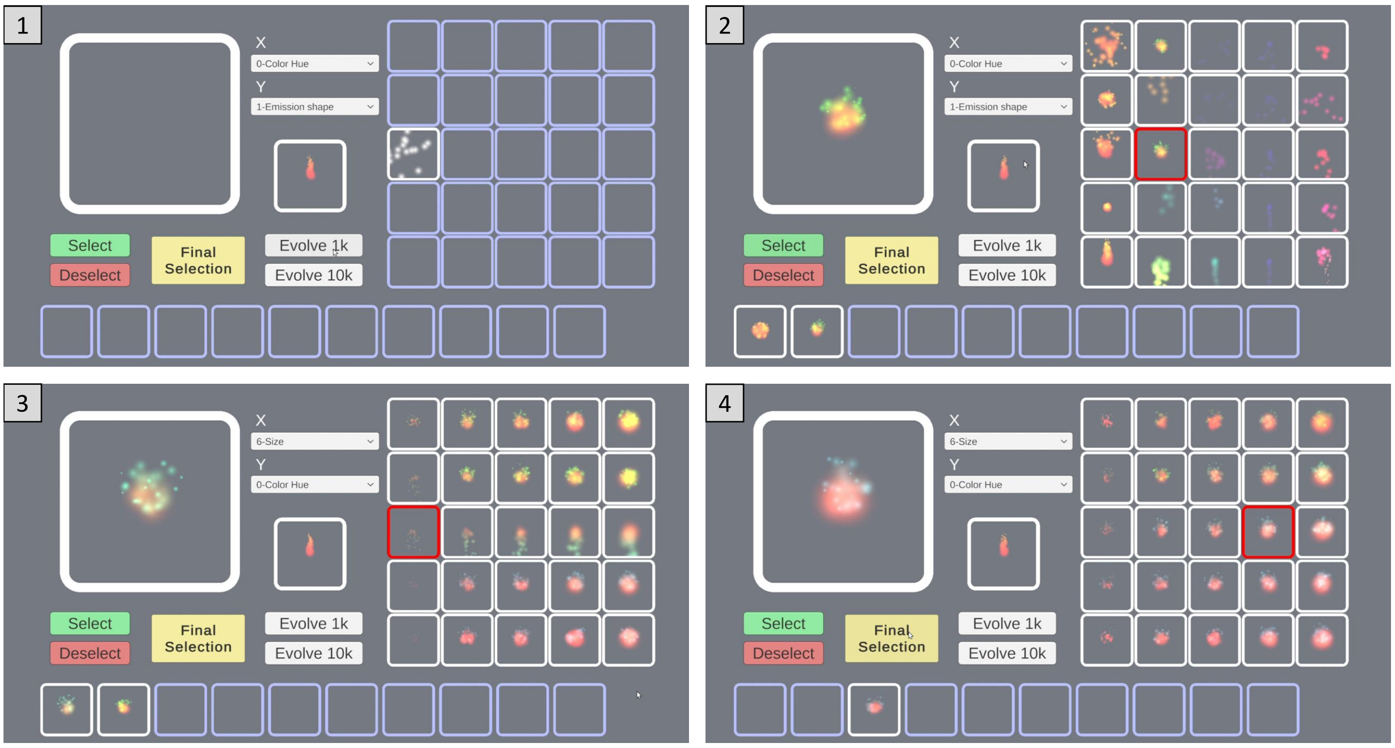

Figure 3: Human-part of the Co-Creation Approach for Particle Systems.

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 quality 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 [35] 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 [35], 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.

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

Table 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.

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 5.2.3.

Fig. 4, shows the scores for the *Fire* PS. We can see that

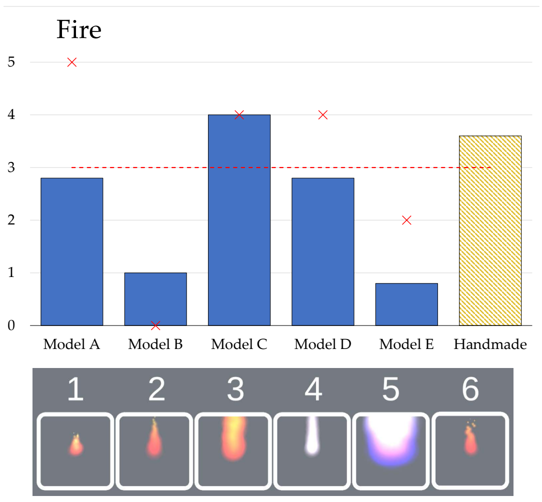

Figure 4: Scoring for each *Fire* PS and a screenshot of the ranking video.

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.

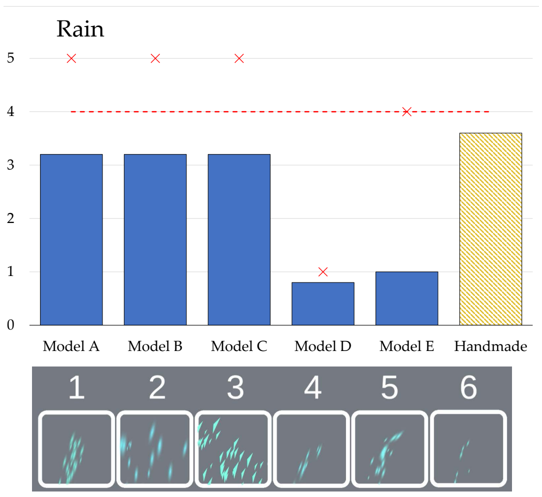

Figure 5: Scoring for each *Rain* PS and a screenshot of the ranking video.

For *Rain*, Fig. 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.

For *Snow*, Fig. 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.

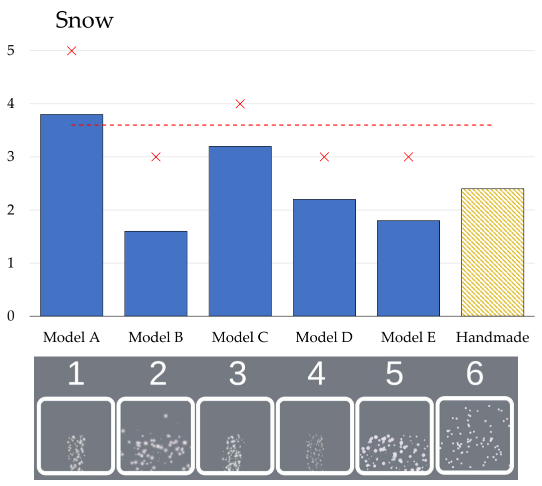

Figure 6: Scoring for each *Snow* PS and a screenshot of the ranking video.

For *Smoke*, Fig. 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.

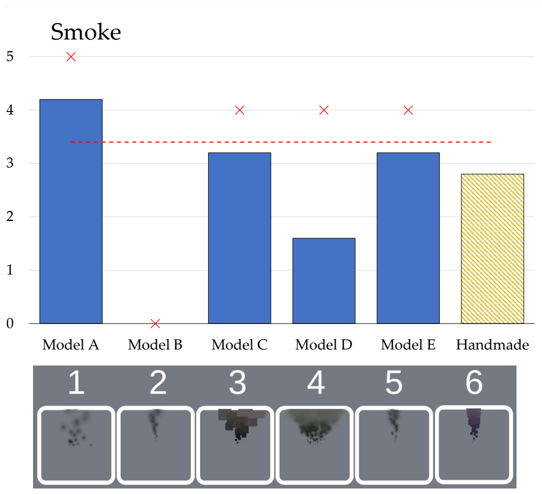

Figure 7: Scoring for each *Smoke* PS and a screenshot of the ranking video.

For *Sparks*, the results favor the handmade PS. 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.

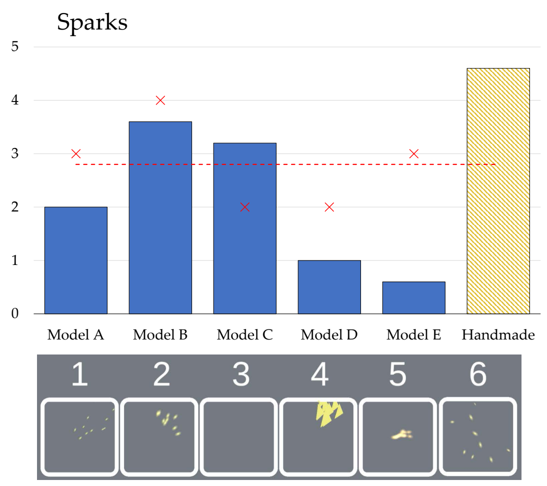

Figure 8: Scoring for each *Sparks* PS and a screenshot of the ranking video.

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 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).

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. [36] and Krueger [37]. 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.

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 [19]. 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 metamodel 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.

7. Threats to Validity

To describe the threats to validity of our evaluation, we have used the classification of Wohlin et al. [38]:

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 [12]:

- 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 machine and human, 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.

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 Busquedas 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).

References

[1] SlashData, "Global developer population report 2019," https://sdata.me/GlobalDevPop19, 2019, [Online; accessed 21-November-2021].

[2] U. Technologies, "Unity real-time development platform — 3d, 2d vr & ar engine," https://unity.com, 2005, [Online; accessed 21-November-2021].

[3] E. Games, "Unreal engine: The most powerful real-time 3d creation tool," https://www.unrealengine.com, 1998, [Online; accessed 21-November-2021].

[4] M. Zhu and A. I. Wang, "Model-driven game development: A literature review," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–32, 2019.

[5] S. Kent, "Model driven engineering," in *Integrated Formal Methods*, M. Butler, L. Petre, and K. Sere, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 286–298.

[6] "Material editor reference, unreal engine 4.27 documentation," https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Materials/Editor/, accessed: 2023-04-05.

[7] "Shader graph, unity features," https://unity.com/features/shader-graph, accessed: 2023-04-05.

[8] N. 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.

[9] D. Floreano and C. Mattiussi, *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press, 2008.

[10] D. Blasco, J. Font, M. Zamorano, and C. Cetina, "An evolutionary approach for generating software models: The case of kromaia in game software engineering," *Journal of Systems and Software*, vol. 171, p. 110804, 2021.

[11] O. M. G. (OMG), "Meta object facility (mof) version 2.4.1," 2013, http://www.omg.org/spec/MOF/2.4.1/.

[12] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.

[13] J. van der Burg, "Building an advanced particle system," https://www.gamedeveloper.com/programming/building-an-advanced-particle-system, accessed: 2023-04-05.

[14] A. Ampatzoglou and I. Stamelos, "Software engineering research for computer games: A systematic review," *Information and Software Technology*, vol. 52, no. 9, pp. 888–901, 2010.

[15] M. Harman and B. F. Jones, "Search-based software engineering," *Inf. and soft.Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[16] W. T. Reeves, "Particle systems—a technique for modeling a class of fuzzy objects," *ACM Transactions On Graphics (TOG)*, vol. 2, no. 2, pp. 91–108, 1983.

[17] J. Tan and X. Yang, "Physically-based fluid animation: A survey," *Science in China Series F: Information Sciences*, vol. 52, no. 5, pp. 723–740, 2009.

[18] B. Zhang and W. 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.

[19] G. Lai, F. F. Leymarie, and W. Latham, "On mixed-initiative content creation for video games," *IEEE Transactions on Games*, vol. 14, no. 4, pp. 543–557, 2022.

[20] M. Charity, A. Khalifa, and J. Togelius, "Baba is y'all: Collaborative mixed-initiative level design," in *2020 IEEE Conference on Games (CoG)*. IEEE, 2020, pp. 542–549.

[21] M. Charity, I. Dave, A. Khalifa, and J. Togelius, "Baba is y'all 2.0: Design and investigation of a collaborative mixed-initiative system," *IEEE Transactions on Games*, 2022.

[22] R. Gallotta, K. Arulkumaran, and L. Soros, "Preference-learning emitters for mixed-initiative quality-diversity algorithms," *arXiv preprint arXiv:2210.13839*, 2022.

[23] E. J. Hastings, R. K. Guha, and K. O. Stanley, “Interactive evolution of particle systems for computer graphics and animation,” *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 418–432, 2008.
[24] ——, “Automatic content generation in the galactic arms race video game,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 4, pp. 245–263, 2009.
[25] J. R. Williams, S. Poulding, L. M. Rose, R. F. Paige, and F. A. Polack, “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.
[26] R. Casamayor, L. Arcega, F. Pérez, and C. Cetina, “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.
[27] W. Kessentini, M. Wimmer, and H. 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.
[28] W. Kessentini and V. 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.
[29] F. Pérez, J. Font, L. Arcega, and C. Cetina, “Empowering the human as the fitness function in search-based model-driven engineering,” *IEEE Transactions on Software Engineering*, vol. 48, no. 11, pp. 4553–4568, 2021.
[30] G. Bavota, F. Carnevale, A. De Lucia, M. Di Penta, and R. Oliveto, “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.
[31] M. Hall, N. Walkinshaw, and P. McMinn, “Supervised software modularisation,” in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 472–481.
[32] R. Tufano, S. Scalabrino, L. Pascarella, E. Aghajani, R. Oliveto, and G. Bavota, “Using reinforcement learning for load testing of video games,” in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2303–2314.
[33] J. Font, L. Arcega, Ø. Haugen, and C. Cetina, “Handling nonconforming individuals in search-based model-driven engineering: nine generic strategies for feature location in the modeling space of the meta-object facility,” *Software and Systems Modeling*, pp. 1–36, 2021.
[34] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.
[35] M. Alvo and L. Philip, *Statistical methods for ranking data*. Springer, 2014, vol. 1341.
[36] J. Kontio, J. Bragge, and L. Lehtola, “The focus group method as an empirical tool in software engineering,” in *Guide to advanced empirical software engineering*. Springer, 2008, pp. 93–116.
[37] R. A. Krueger and M. A. Casey, *Designing and conducting focus group interviews*. Citeseer, 2002, vol. 18.
[38] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.