Procedural Content Generation via Knowledge Transformation (PCG-KT)

Anurag Sarkar, Matthew Guzdial, Sam Snodgrass, Adam Summerville, Tiago Machado, Gillian Smith

Abstract—We introduce the concept of Procedural Content Generation via Knowledge Transformation (PCG-KT), a new lens and framework for characterizing PCG methods and approaches in which content generation is enabled by the process of knowledge transformation—transforming knowledge derived from one domain in order to apply it in another. Our work is motivated by a substantial number of recent PCG works that focus on generating novel content via repurposing derived knowledge. Such works have involved, for example, performing transfer learning on models trained on one game's content to adapt to another game's content, as well as recombining different generative distributions to blend the content of two or more games. Such approaches arose in part due to limitations in PCG via Machine Learning (PCGML) such as producing generative models for games lacking training data and generating content for entirely new games. In this paper, we categorize such approaches under this new lens of PCG-KT by offering a definition and framework for describing such methods and surveying existing works using this framework. Finally, we conclude by highlighting open problems and directions for future research in this area.

Index Terms—procedural content generation, transfer learning, conceptual blending, computational creativity, transformation

I. Introduction

Procedural content generation (PCG) describes the set of approaches for generating game content via algorithmic means. There are several lenses through which we describe and understand PCG including search-based PCG (SBPCG) [1], PCG as a constructive method [2], PCG as a representation of values [3], and PCG as a means for providing different player experiences [4]. Each of these lenses has been informed primarily by a focus on formalizing and generating content for a single game.

A recent interest in the application of machine learning to procedural content generation (PCGML [5]) has revealed not only a new technical approach to content generation but also new research questions and trajectories informed by a data-oriented approach to content generation. PCGML approaches generally focus on learning a distribution over existing content

A. Sarkar is with the Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA (email: sarkar.an@northeastern.edu)

M. Guzdial is with the Department of Computing Science, Alberta Machine Learning Institute, University of Alberta, Edmonton, AB, Canada (email: guzdial@ualberta.ca)

S. Snodgrass is with modl.ai, Copenhagen, Denmark (email: sam@modl.ai)

A. Summerville is with the College of Science, California State Polytechnic University, Pomona, CA, USA (email: asummerville@cpp.edu)

T. Machado is with Northeastern University, Boston, MA, USA (email:tiago.la.machado@gmail.com)

G. Smith is with the Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA, USA (email:gmsmith@wpi.edu)
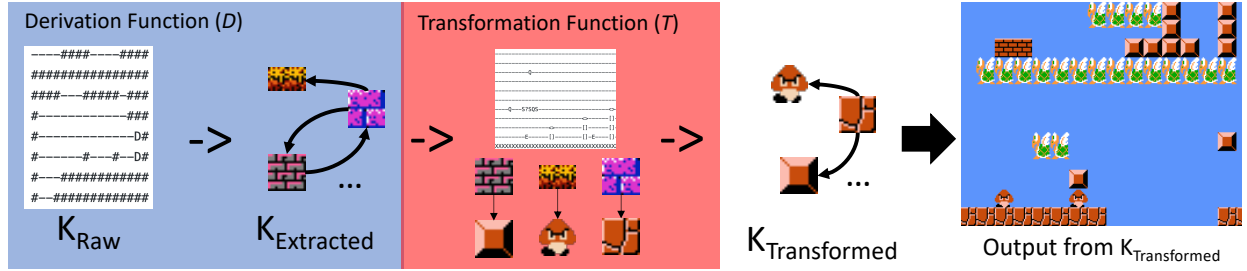
and then sampling from that distribution to produce new content. However, PCGML models can only generate content from within a distribution learned from their training data. This has led to certain core problems in PCGML namely, how do we create new game content for a game with little or no training data and how can we produce entirely new content that references existing content but that is distinct from it? Such problems give rise to further questions and avenues of work. For example, generating new content that inherits important characteristics from existing content without directly repeating or plagiarizing it, or recombining knowledge derived from existing content to generate new content and discover new game and level designs.

We identify a number of potential, longstanding challenges in PCG that can be addressed by answering the above questions. One is automated game design i.e., the production of entirely novel video games. While there have been prior approaches to this problem [6], they are still relatively few in number. This is in part due to the large amount of design knowledge required to produce such a system [7]. If we could transform knowledge from existing games towards the production of new games, we could sidestep this issue.

Controllability stands as another core PCG problem, with the focus on how users might best achieve their desired goals with a PCG system. A common problem when it comes to controllability is that a procedural content generator is unable to meet a user's needs [8]. For example, a user may wish to generate some content X. If the closest the PCG system can get is to generate some content Y, then the user will be unable to satisfactorily control the system. But if the system had the ability to transform $Y \rightarrow X$, then this problem could be resolved.

In this article, with a view towards addressing such problems, we describe a new lens for characterizing PCG research and practice: Procedural Content Generation via Knowledge Transformation (PCG-KT). We provide a framework and vocabulary for describing PCG systems by the methods and extent to which they transform knowledge between domains. We further survey existing systems that are situated in novel positions within this framework, and pose new potential areas of PCG research that are uncovered by this framing.

We would like to note that we introduce PCG-KT as a lens for understanding PCG processes, not as a category of PCG approaches. Unlike SBPCG or PCGML, we are not identifying a group of PCG approaches related by an underlying AI technology. Instead, we use this lens and framework to allow us to compare and contrast PCG approaches based on the extent to which they involve the transformation of knowledge. All PCG

Fig. 1: Example of our framework applied to an example from [9]. Initial *Kid Icarus* level data (raw knowledge $\mathbb{G}_1$) is used to derive a Markov level design model (extracted knowledge $\mathbb{K}_1$) via a derivation function (D). This knowledge is passed as input through a transformation function (T), which uses a mapping and a Mario level to derive a Markov level design model for *Super Mario Bros.* levels (transformed knowledge $\mathbb{K}_2$), which can be used to output novel content.

approaches could be understood as falling within PCG-KT, but most involve zero or trivial transformation. On the other end of the spectrum, there are approaches that radically transform the derived knowledge to approximate secondary or nonexistent knowledge. For clarity in introducing these concepts, we focus on the latter of these in this paper. However, we view PCG-KT as a descriptive framework, not as a prescriptive set of conditions determining sufficient knowledge transformation to qualify as PCG-KT.

II. PCG VIA KNOWLEDGE TRANSFORMATION

In this section, we outline nomenclature and a generalized framework for PCG via Knowledge Transformation (PCG-KT). We acknowledge that PCG is broad and that there are many different kinds of content, knowledge, and transformation processes used in the field. We refer the reader to Section III for a framework for differentiating between types of knowledge transformation, as well as examples of how existing systems fit into this framework.

We provide a formalized definition for PCG-KT in order to identify prior work, to prompt reflection on how this lens both fits and reframes existing approaches to PCG, and to encourage future work in under-explored directions.

For our definition we employ the following vocabulary:

- **Games** ($\mathbb{G}$: a set of games; G: a game $G \subseteq \mathbb{G}$). A game, or set of games, forms the domain that situates and contextualizes the knowledge transformation process. We deliberately do not adopt any one prescriptive definition of "game" here: a game might be theoretical, in-development, or published; renowned or unknown; a research "toy" or established; digital or analog; procedurally generated or fully authored by a human. The nature of the game(s) informs the types of knowledge that can be derived, and the broader context and cultural situation of that knowledge.
- **Knowledge** ($\mathbb{K}$: a set of knowledge derived from a set of games; K: a unit of knowledge). We use the term "knowledge" to describe information derived from a set of games that can be applied to a new set of games (with or without transformation). This information could take many different forms, including authored content (sprites, level maps), functional design aspects (e.g., mechanics, game states), or broader information related

to overall game experience (e.g., inferred style heuristics, automated playtraces). The knowledge may vary in scope (e.g., a single sprite or an entire ruleset). It may also be represented in a form that is independent of what either the original or target games use to function. For example, the ASCII representation used in level maps in the Video Game Level Corpus (VGLC) [10] is knowledge that is derived from those games, yet not in the format those games use (and indeed, elides some features of the original level maps such as non-interactable decorations).

- **Derivation** (A derivation function D that produces knowledge from a game: $D(\mathbb{G}_i) = \mathbb{K}_i$). Knowledge should be derived from some set of games according to a reproducible process. This knowledge may be derived by hand, automatically, or via some hybrid process. The derivation process does not need to apply directly to the game as an artifact (e.g., spritesheets extracted from game ROMs). For example, knowledge derived through inference from playtraces or gameplay video still follows a clear derivation process related to the game.
- **Transformation** (A transformation function T that takes knowledge as input and produces knowledge as output: $T(\mathbb{K}_i) = \mathbb{K}_j$). This function accepts knowledge that has been derived from a set of games and alters it in such a way that it instead represents knowledge that could be derived from a new set of games. Note that this could be an identity transformation if the target and input games are sufficiently similar (e.g., extracted jumping mechanic information shared between platforming games.)

PCG via Knowledge Transformation means casting the generative process as one of deriving, transforming, and applying knowledge from one (set of) games to another (set of) games. More formally:

$$\mathbb{K}_1 = D_1(\mathbb{G}_1) \tag{1}$$
$$T(\mathbb{K}_1) = \mathbb{K}_2 \approx D_2(\mathbb{G}_2) \tag{2}$$
$$\mathbb{G}_2 \not\subseteq \mathbb{G}_1 \tag{3}$$

An overview of the application of this PCG-KT framework is depicted in Figure 1.

III. FRAMEWORK FOR DESCRIBING PCG VIA KNOWLEDGE TRANSFORMATION

In this section, we propose a set of features and axes for describing approaches to PCG-KT. Using the definitions presented in the previous section, we can frame many existing PCG approaches as exhibiting a form of knowledge transformation. For clarity, we will discuss a higher level descriptive category and then its underlying features and possible axes.

A. Knowledge Structure

When discussing and describing these transformation approaches, it is important to consider the *structure* of the knowledge being transformed. Specifically, we consider structure at each of the three different stages of the transformation process; the raw input, the extracted input, and the transformed output.

1) Raw Knowledge: This is the knowledge closest to the input domains, meant to represent the input game(s) or domain(s) themselves, and is denoted:

$$\mathbb{G}_i$$

Its structure depends on the context of the transformation, especially the domain (i.e., which game(s)); and the knowledge to be extracted from that domain (e.g., level designs, story structures, sprite palettes, character designs, etc.). For example, in a lot of PCGML work, a tile-based representation of a level can be taken as the raw knowledge, as in Figure 1. This enables methods for learning about the level structure in discrete patterns (e.g., [11], [12]). Alternatively, if the process started from images of input levels and converted them into some ML-parsable representation like tiles, the input images would be considered the raw knowledge. Other raw input formats such as segmented videos of playtraces enabled other methods to learn about gameplay relationships and object placements in the levels [13]. This is relevant to knowledge transformation as all later steps, including the possible output of the transformation, will depend upon this knowledge. Thus, different raw knowledge formats have different affordances for a PCG-KT process. For example, the tile-based raw knowledge would not support working with portals as it does not include them.

2) Extracted Knowledge: This is the knowledge that is immediately extracted from the raw input knowledge prior to performing transformation, and is denoted:

$$\mathbb{K}_i = D_i(\mathbb{G}_i)$$

Its structure depends on the raw input knowledge, the methods used for extracting the knowledge, and the type of transformations to be performed. Such knowledge can be categorized based on whether it is direct or latent. Closer to the direct end, we could imagine a multi-layer [14] or hierarchical [15] representation of the levels annotated with level features. Somewhere in the middle, we could have a learned probability table of tile distributions [11]. At the latent end, we could have the weights of a neural network trained on those levels [12] or the latent vector representations of the levels [16], [17].

3) Transformed Knowledge: Lastly, we have the knowledge after the transformation has been performed, which is denoted:

$$T(\mathbb{K}_i) = \mathbb{K}_j \approx D_j(\mathbb{G}_j)$$

The structure of this knowledge can be described similarly as the structure of the knowledge prior to transformation, and so we will not repeat the same descriptions. It is important to note however that while many existing approaches use the same structure for the extracted and transformed knowledge, this is not required. For example, we can conceive of a method that takes a set of learned probability tables that model the level design, and outputs a set of latent vectors meant to blend those models together, or that takes a set of neural network weights from two models trained to produce different kinds of game content, and outputs content for a third, unseen game.

B. Derivation Function

The derivation function is the process by which the extracted knowledge is derived from the raw knowledge described above. In the knowledge extraction equation $\mathbb{K}_i = D_i(\mathbb{G}_i)$, the derivation function is D_i. There are a variety of ways to construct the derivation function. We describe three broad categories ranging from hand-authored to hybrid to automated, representing a sliding scale where the function can be either more or less hand-authored (or automated), with fully manual and fully automated lying on the extreme ends of the spectrum.

1) Hand-Authored: Hand-authored derivation functions rely on manual authorship and domain expertise to pull the extracted knowledge from the raw knowledge. An example of such a function would involve a human domain expert analyzing a set of input levels and then devising a set of design rules for the given game (e.g., holes can only be 4 tiles wide; cave levels can only have specific enemies, etc.). This extracted knowledge can then be used in the transformation. Another example could be manually tagging and labelling structures or patterns within content (levels, stories, quests, etc.).

2) Automated: At the other end of the spectrum, automated derivation functions rely on automated rules, heuristics or machine learned information in order to pull the extracted knowledge from the raw knowledge. For example, Summerville and Mateas [12] and Snodgrass and Ontañón [11] use backpropagation and occurrence counting, respectively, as the derivation function for pulling out the extracted knowledge. In those examples, the extracted knowledge is the trained model, be it a conditional probability distribution or an LSTM. If we think of the domain of stories or quests, we can imagine an approach that pulls out word embeddings from the narrative text to use as the extracted knowledge. The derivation function here could be a simple word clustering method, or a neural network approach that returns latent vector encodings.

3) Hybrid: These leverage both automated methods and human expertise. Most approaches in practice are likely to fall under this category, as even automated approaches may rely on domain expertise for setting up rules, training schema, labels, etc. Consider the automated approach of extracting a conditional distribution of tile types to model design knowledge from a game level. We can view this as a hybrid

approach when the user is defining the neighborhood of tiles the model should be looking at for extracting the distribution. Alternatively, if we consider the hand-authored approach of tagging structures or patterns in stories, then the initial output of this approach could be used as the starting point for training a more automated approach to tag new examples.

C. Transformation Context

The *context* of the transformation or how the transformation knowledge is going to be used is another important factor to consider. For describing the context, we propose the following axes, though these are not meant to be exhaustive.

1) Novelty: This describes if the transformation is intended to approximate an existing domain or explore new domains. For example, on the approximation end of the axis, Snodgrass and Ontañón [9] try to learn a mapping of level representations from one game to another existing game, where some prior knowledge is known. On the other end, Guzdial and Riedl [18] present an exploration approach where knowledge extracted from a set of input games is transformed and recombined to create entirely new games. Approaches can be placed along this spectrum based on how much prior knowledge of the approximated domain is known, and the extent to which the explored domains are different from the input domains.

2) Usage: This aspect of the context relates to the use of the approach and is less of an axis and more a categorical description of where the approach sits in a user context. For example, is the approach being used in a prototyping setting to explore new game ideas, a research environment to test the underlying methods, or to produce a complete new game for player consumption? Knowing this aspect of the context can directly influence what kinds of knowledge can be used and what the transformation function can look like. Usage in this case is distinct from novelty, as for the same usage or goal, we might have different values of novelty. When prototyping new game mechanics, we might want very novel mechanics (i.e. distinct from all prior mechanics our system has seen) or we might more about how well the mechanics fit with other aspects of the design (e.g., narrative, aesthetics, etc.)

D. Transformation Properties

This is a broad category that encompasses many aspects of the transformation performed and is targeted towards describing the extent to which knowledge is transformed and the relationship between the transformed and input knowledge. This is distinct from just identifying input and output knowledge as in the definition, where we did not explicitly outline their possible relationships. When describing the extent of the transformation, we need to consider the distance of the transformed knowledge from the input knowledge. We use the term *distance* rather loosely. Instead of defining a specific distance metric, we discuss two types of distance depending on the structure of knowledge and the properties of interest.

1) Representational Distance: This refers to the difference between the knowledge structure in the input and transformed knowledge. For example, the representational distance is quite low if interpolating between two latent vectors as all the

knowledge has the same encoding. However, this distance is much higher when, for example, transforming a story graph into a set of three dimensional points in a game world. Notice, the representational distance for many approaches is fairly low as the knowledge structure is typically the same for the input and transformed knowledge (e.g., mapping one tile or sprite set to another, or modifying a learned distribution).

2) Content Distance: This is the difference between the content of the input and transformed knowledge. For example, the content distance between sets of sprites extracted from two different Mario levels is much lower than the content distance between the sets of sprites extracted from a Mario level and a Mega Man level. If we consider two transformations, one that produces a new set of sprites by combining individual sprites from Mario and Mega Man, and one that creates a new set of sprites by randomly choosing a subset of sprites from Mario and Mega Man, then the content distance between the resulting blended sprite set and the input set would be larger than the distance between the resulting randomly joined sprite set and the input set. We can think of this as equivalent to the amount of shared information between the input and transformed knowledge, regardless of how that information is represented. As an example, Sarkar et al.'s approach [17] of learning blended latent representations of several input games has a higher content distance than Snodgrass and Sarkar's approach [14] that generates new levels by combining existing parts of existing game levels using binary space partitioning, which in turn has higher content distance than prior PCGML methods that operated on single-game domains.

Note that these two distances are independent of each other. Two sets of knowledge containing the same information but represented in different ways would have high representational distance but low content distance (e.g., representing level design knowledge for a game as tile-based representations of a set of levels vs. describing the levels as a set of rules for placing objects and enemies). Similarly, two sets of knowledge can represent different information in the same way. For example, two neural networks with the same architecture trained on two different games would result in knowledge sets with low representational distance, but high content distance as the weights of the networks would differ.

E. Transformation Function

The *function* used to perform the transformation relates to all other aspects of PCG-KT since it relies on the *context* of the transformation, needs the various knowledge *structures* in order to perform the transformation and the *properties* of the transformation depend upon this function. Here we break down the various ways of describing the function.

1) Approach: This refers to the actual procedure or manner in which the transformation is being performed. This can be through updating a probability table [11], backpropagation through a neural network [12], iteratively growing a graph from existing extracted graphs [18], interpolating between latent vectors [16], [17], or reducing detailed knowledge representations from various domains into a single uniform representation [14], [17], to name a few. For this category, the

		Game Blending	**Conceptual Expansion**	**Domain Transfer**
Knowledge Structure	*Raw Knowledge* *Extracted Knowledge* *Transformed Knowledge*	VGLC Levels VAE latent vectors Blended VAE latent vectors via interpolation and labeling	Gameplay Videos Game Graphs Conceptual Expansion of Game Graphs	VGLC Levels MdMC Conditional Distribution VGLC Levels Converted to Target Domain
Derivation Function	*Type* *Description*	Automated Training via Backpropagation	Automated Rule Learning and Probability Estimation	Automated Estimating Tile Distributions via Observation Counting
Transform Context	*Novelty* *Usage*	New Research Environment	New Research Environment	Transfer Research Environment
Transform Properties	*Representation Distance* *Content Distance*	Low High	Low High	Low Medium
Transform Function	*Approach* *Input Domain* *Output Domain* *Cardinality*	Interpolation/evolution of VAE latent vectors Set of N games New game blending the N games Many-to-One, Many-to-Many	Conceptual Expansion Set of N games New game represented as a combination of the N games Many-to-One, Many-to-Many	Multi-stage Pipeline for Defining Mapping between Tiles Target Game and Input Game(s) Input Game(s) Transformed to Target Game Representation One-to-one, Many-to-one

TABLE I: The PCG-KT framework applied to three techniques: game blending, conceptual expansion and domain transfer.

function used can be nearly any procedure already used in the PCG, ML, creativity, design, or AI literature, including manual and hybrid approaches. Explicitly identifying the procedure where the transformation takes place can help to more clearly delineate the transformation approach as a whole.

2) Input/Output Domains: The input and output domains of transformation help characterize the nature of the transformation process. For example, in works centered around blending different games, the input domain is the set of games to be blended and the output domain is the final blended game. Thus, here the transformation process is one of transforming input knowledge extracted from the individual games into output knowledge recombining properties of the input games together. Similarly, in the case of domain transfer for level design, the input domain consists of a set of one or more input games and a target game and the output domain consists of levels of the input game(s) transformed to the target game representation.

3) Cardinality: This refers to the number of input sources and number of outputs. We consider four types of cardinality:

1) *One input domain to one output domain*: this is a standard knowledge transfer where knowledge from one domain is transformed to represent knowledge from another domain [9].
2) *Multiple inputs to one output*: this is typically a blending approach where the input domains are transformed and combined in some way to get a different output domain (either new or existing) as in [14], [17], [18].
3) *Multiple inputs to multiple outputs*: this can also be blending or approximating, but likely in a more exploratory way where the input knowledge is transformed and recombined in various ways to explore different output domains. Examples of this are [19], [20] where the input is a group of games but rather than produce a single blended output, different labels are used to

produce different combinations of blends where each unique label configuration can be thought of as mapping the set of input games to a separate blended domain made up of a unique combination of the input games.
4) *One input to multiple outputs*: this is also a more exploratory transformation where the knowledge from a single domain is transformed and possibly abstracted in order to explore multiple output domains.

To the best of our knowledge, while there have been systems that combined multiple inputs into multiple outputs consecutively, there have not been any systems that have attempted to simultaneously generate multiple outputs dependent on one another (e.g., a series of games, a set of sprites, or a soundtrack).

To illustrate the framework using concrete examples, Table I depicts the result of applying the PCG-KT framework to three separate applications, game blending [16], conceptual expansion [18], and domain transfer [9], showing how the features and axes discussed above can be categorized in each case. The raw knowledge $\mathbb{G}_i$ consists of VGLC levels for game blending and domain transfer while consisting of gameplay videos in the case of conceptual expansion. In all three cases, the derivation function ($D_i(\mathbb{G}_i)$) involves a machine learning algorithm (such as a neural network, Bayes net, or Markov model) which is used to obtain the extracted knowledge ($\mathbb{K}_i$). This is in the form of latent vectors for game blending, game graphs for conceptual expansion, and conditional distribution tables for domain transfer. For the transformation function ($T(\mathbb{K}_i)$), conceptual blending employs latent vector interpolation, whereas conceptual expansion requires a multi-stage pipeline including defining tile mappings to obtain the final transformed knowledge $\mathbb{K}_j$. This consists of blended latent vectors for game blending, a recombined game graph representing a new game

in the case of conceptual expansion, and levels in the target domain for domain transfer. These applications leverage two broad categories of transformation functions—combinational creativity and transfer learning. These and the examples in Table I are discussed in more detail in Section V.

IV. RELATED WORK

In this section, we discuss related areas of prior work and compare them to the focus of this paper.

A. Procedural Content Generation

PCG-KT is a framework for thinking about a particular PCG process, thus all prior PCG work represents a related area.

One clear area of related work is Procedural Content Generation via Machine Learning (PCGML) [5], given that we identify PCG-KT as a solution to major problems with this approach. However, we believe PCG-KT also applies to cases where the input knowledge is hand-authored, as in the work of Permar and Magerko [21]. Thus we identify PCG-KT as applying to PCGML, but not wholly subsumed by it.

Supervised PCGML attempts to take in knowledge in the form of existing content and transform it (via training an ML model and then sampling from that model) into additional content. However, we do not consider this process to be PCG-KT in most cases due to a difference in goals. For example, a PCGML approach might attempt to learn a model of level design that allows for the generation of content (e.g., new levels) that would fit some existing game (e.g., Super Mario Bros.) [12], [22]. In this case, the goal is explicitly that the set of output levels should substantially resemble the original knowledge. This is typically done by learning a distribution based on the original, existing content and then sampling from that distribution. It follows then that any output from this learned model represents the same knowledge from the original game $\mathbb{G}$. While our definition of PCG-KT covers the case in which the input game matches the output target game, this represents a minimal or trivial transformation.

We also identify PCG-KT as distinct from most search-based PCG (SBPCG) approaches, such as genetic algorithms or evolutionary search [1]. In these approaches, we can consider this search space as having been defined by the author's own game design knowledge. Thus, it is derived from a theoretical set of games $\mathbb{G}$, in this case all of the games the author considers when they define the search space. It then follows that any output from this search space represents knowledge from $\mathbb{G}$. There is prior work where an entire playable game is produced via SBPCG [6]. However, even in these cases, that game already existed in the unchanged search space. Thus, the output games must necessarily exist in $\mathbb{G}$. In other words, because the author's definition of a space of games included the output game, there is no transformation that has occurred and the knowledge remains unchanged. Note however that search-based methods could be classified as PCG-KT if the search is used to perform knowledge transformation. The appropriateness of applying PCG-KT as a lens is dependent on the purpose of a system, not the system's technical approach.

B. Design Patterns

In software engineering, the design structures that make software more flexible and reusable are known as design patterns [23]. The use of design patterns spans several disciplines and in general, a design pattern describes a recurrent problem in an environment and maps it to a core of solutions that can be reapplied and adapted without needing to be reinvented from scratch [24]. Because the design of a game involves activities that are not related entirely to problem solving and engineering skills, the concept of game design patterns was introduced to address game mechanics, game design, and game art [25], [26]. Prior works have used game design patterns to analyze level designs [27], [28], [29] and interactive narrative [30]. Björk and Holopainen [25] defined game design patterns as pieces of game design knowledge which could be identified when they occurred in multiple games (e.g., a companion character or collectible). There has been work that takes these design patterns and attempts to apply them to new games [31]. However, we note that in these cases there is no effort made to transform these patterns outside of their initial identification. Thus a single design pattern applied to some new game is understood to be the same pattern as that in the original games.

C. Transfer Learning and Domain Adaptation

In transforming knowledge from one domain to another, PCG-KT touches upon transfer learning and domain adaptation and in later sections, we will see examples of PCG-KT work that leverages these techniques. Zhuang et al. [40] define transfer learning methods as those that attempt to leverage knowledge learned for a source task and use this source knowledge to benefit the learning of knowledge for a target task. Domain adaptation is defined by Pan et al. [41] as the method of solving a learning problem in a target domain by using training data in a different but related source domain. In games, other than the works discussed in this paper, such methods have primarily been used for transferring the behavior of agents from one environment to another. Chaplot et al. [42] used deep Q-networks to transfer agent navigation behavior from seen to unseen maps in the VizDoom environment. Note that this does not fall under PCGKT since the knowledge transfer happens within the same game. Consequently, no content is generated for any new game. Rusu et al. [43] introduced progressive networks for transfer learning agent behavior across different Atari games while Kansky et al. [44] used schema networks to learn basic physics models of entities, which were then used in novel configurations of Breakout not seen during training. Melnik et al. [45] learned modules such as object detection and trajectory based physics to transfer an agent policy from Pong to Breakout. Relatedly, several approaches have been attempted to ease the ability to transfer knowledge learned in one game and apply it to another. Mittel and Munukutla [46] and Gamrian and Goldberg [47] handle this task by disentangling the visual aspects from the control aspects of a learned agent. [46] transfer agent policies from Atari Pong to Breakout while [47] perform transfer learning to build agents capable of playing Breakout and Road Fighter. However, note that in each of [43]-[47], while some knowledge

Transformation Function		Papers	Method	Generated Content
Main Category	Secondary Category			
Combinational Creativity	Conceptual Blending	[13]	Hierarchical Bayes Network	Levels
		[16], [17]	Variational Autoencoder (VAE)	Levels
		[32]	Variational Autoencoder (VAE)	Jumps
		[19], [20]	Conditional VAE	Levels
		[33]	Graph Structure Knowledge Base	Games
		[34]	Manual Conceptual Blending	Games
	Conceptual Expansion	[35]	Bayesian Networks	Levels
		[7], [18]	Probabilistic Graphical Models	Games
Transfer Learning	Domain Adaptation	[9]	Multidimensional Markov Chains	Levels
		[36], [14]	VAE, Binary Space Partitioning	Levels
	Style Transfer	[37]	Tile-to-Affordance Mapping Autoencoder/Markov Model	Levels
	Design Patterns	[38]	Case-based Reasoning	Levels
	Data Mining Association	[39]	Apriori Algorithm	Game Rules

TABLE II: Summary of the PCG-KT works discussed in Section V.

concerning agent behavior is being transferred from one game to another, no new content is created as a result of the transfer, thus failing to fall under the PCGKT definition. In [48], Kim et al. introduced GameGAN, a neural network that can learn a model of a game by separately modeling the static and dynamic elements. This allows swapping the static and dynamic elements of different games to perform a sort of transformation. However, the transformation is minimal, closer in nature to swapping out sprites or background images than a transformation of game structure or mechanics.

D. Computational Creativity

Computational Creativity (CC) is the umbrella term for efforts to reproduce human creativity in computers [49]. Since we focus on how knowledge related to PCG can be transformed to produce novel output in a particular creative process, it relates to CC. CC has long been linked to games [50], however the majority of this work has focused on "exploratory" creativity, which is strongly associated with SBPCG discussed previously. A more related type of creativity to PCG-KT is "transformational" creativity [51], where a conceptual space, which can be thought of as analogous to a search space or distribution, is transformed via an alteration in the variables representing the space. This has strong parallels to our PCG-KT definition, though transformational creativity is typically much more loosely defined. Boden defined a third type of creativity: combinational creativity [51]. This type of creativity refers to the combination of concepts, and has inspired a large number of computational processes to attempt to replicate it, with the most famous being conceptual blending [52]. This approach is known in the psychology field as conceptual combination [53], and has been studied for decades as a ubiquitous cognitive process. One of the two categories we introduce for transformation functions is based on combinational creativity. There are other CC methods that have some overlap with PCG-KT. For example, novelty search [54] and surprise search [55], which attempt to locate members of a search space or environment that are novel and surprising respectively. While this might seem to be equivalent to PCG-KT, because the search space remains unchanged (the knowledge it represents remains untransformed), we do not
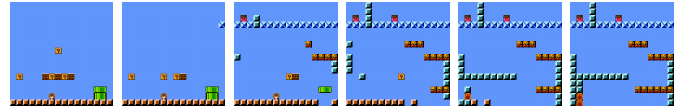
Fig. 2: Example of blending between *Super Mario Bros.* and *Kid Icarus*. Reproduced with permission from [16].

consider it to be the same process. However, we could imagine employing approaches like novelty or surprise search towards PCG-KT, seeking novel or surprising transformations, and thus we consider these and other CC methods to be complementary to those discussed in this paper.

V. CATEGORIES OF TRANSFORMATION FUNCTIONS

In this section, we survey existing literature that fits under the banner of PCG-KT. While we introduced a definition of PCG-KT earlier and several aspects to consider when designing a PCG-KT system, we anticipate that the primary differentiator of PCG-KT systems is their transformation functions: the process that actually changes the knowledge. Thus, we identify two broad categories of PCG-KT transformation functions: (1) combinational creativity and (2) transfer learning. In the first, the transformation is based on combining existing PCG knowledge to produce other PCG knowledge capable of generating novel content. In the second, knowledge is transformed via the use of transfer learning [40], taking knowledge from a source domain and altering it to better fit some target domain. These roughly correspond to many-to-many and many-to-one cardinality for combinational creativity and one-to-one cardinality for transfer learning. We discuss both categories and cover any subcategories and particular recent approaches that fall into each category. To help situate each discussed work in the PCG-KT framework, we highlight the form of the input knowledge and that of the output knowledge obtained after applying the transformation process. Here, input and output knowledge refer to the input extracted knowledge and output transformed knowledge of the transformation function ($T(\mathbb{K}_i) = \mathbb{K}_j$) described previously. An overview of the discussed works is given in Table II.
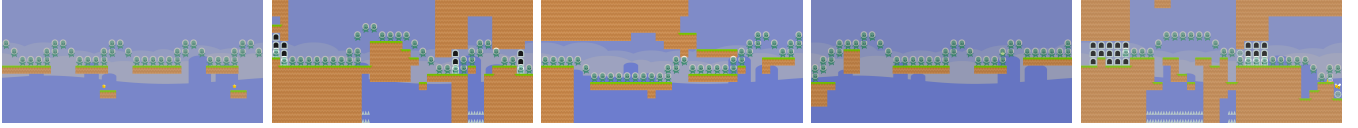
Fig. 3: Example interpolation between *Mega Man* and *Metroid*. Reproduced with permission from [17]

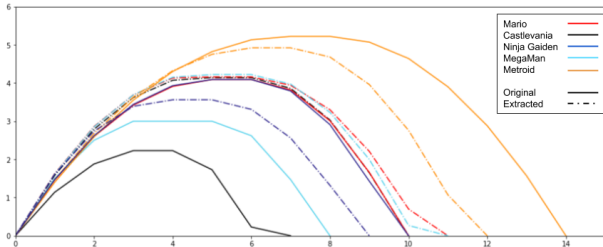Fig. 4: Example output of a blended level design model.

Fig. 5: Original game jumps (solid) vs generated jumps (dash-dotted) extracted from blended domains. Reproduced with permission from [56].

A. Combinational Creativity

Combinational creativity is a type of creativity that deals with the creation of new artifacts, ideas and/or knowledge via recombining existing knowledge in novel, unfamiliar ways [57]. Guzdial and Riedl drew on combinational creativity in an early argument as to how to solve the problem of producing novel content with supervised PCGML [35]. There are a great number of approaches to computationally represent combinational creativity. These approaches include amalgamation [58] and compositional adaption [59], but these approaches have largely not been applied to PCG problems. Thus we focus on two specific combinational creativity approaches: conceptual blending and conceptual expansion.

1) Conceptual Blending: Conceptual blending is a combinational creativity technique focusing on producing novel concepts by combining or blending elements from existing concepts. Fauconnier and Turner [52] formalized conceptual blending in their 'four space' theory which describes a concept blend as comprising of four spaces - two *input* spaces consisting of the concepts to be blended or combined, a *generic* space into which the two input concepts are projected in order to find equivalence points amenable for blending, and a *blend* space into which these points of equivalence are projected and from which novel concepts and patterns are evolved. In the following sections, we review prior work that utilizes conceptual blending and how they can be specified under the PCG-KT framework.

In [13], Guzdial and Riedl employed conceptual blending on learned level design models. Here, the input knowledge consisted of level design models represented as hierarchical Bayesian networks, thus they were graph-like in nature. This

made it simple to adapt the traditional conceptual blending algorithm to these models. The output knowledge was produced by blending the models to produce new level design models that could in turn generate new kinds of levels. For example, combining two generative level design models that produced underwater and castle Mario levels to create a model that could generate underwater-castle levels as in Figure 4.

In [33], Nelson and Mateas outlined an approach to recombine game mechanics in order to generate new games. The generation process involved taking a verb and/or noun from users to describe a theme and generating an appropriate game in response, using a set of pre-defined abstract game templates, each of which could be implemented using different stock sets of mechanics. The input knowledge here thus consisted of the game templates and mechanic sets selected for combination based on user-supplied verbs and nouns and the output knowledge was the resulting generated game produced via combining the selected template and mechanics.

In [34], Gow and Corneli proposed generating entire games by blending elements from existing ones. In their framework, the input knowledge comprised of game specifications in the Video Game Description Language (VGDL) [60]. As an example, they manually applied conceptual blending as the transformation function on the VGDL specifications of the games *Frogger* [61] and *The Legend of Zelda* [62]. This produced as output knowledge the VGDL specification for *Frolda*, a new game combining *Frogger's* mechanic of crossing a road full of hazards with *Zelda's* lock-and-key mechanic.

This framework has been operationalized through the use of ML in works by Sarkar and Cooper [16] and Sarkar et al. [17]. Both use variational autoencoders (VAEs) [63] which consist of encoder-decoder networks that learn continuous, latent representations of the input data. By training on levels from multiple games, the learned VAE latent space spans all the games used in training and enables sampling blended levels. In these works, the input knowledge is comprised of the learned latent representations extracted from the original game levels (taken from *Super Mario Bros.* [64], *Kid Icarus* [65], *Metroid* [66], *Mega Man* [67], *Castlevania* [68] and *Ninja Gaiden* [69]) along with the mapping between levels and latent vectors learned by the VAE encoder and decoder. The output knowledge is composed of the latent representations of blended levels obtained by applying conceptual blending via interpolation between latent vectors or by evolving such
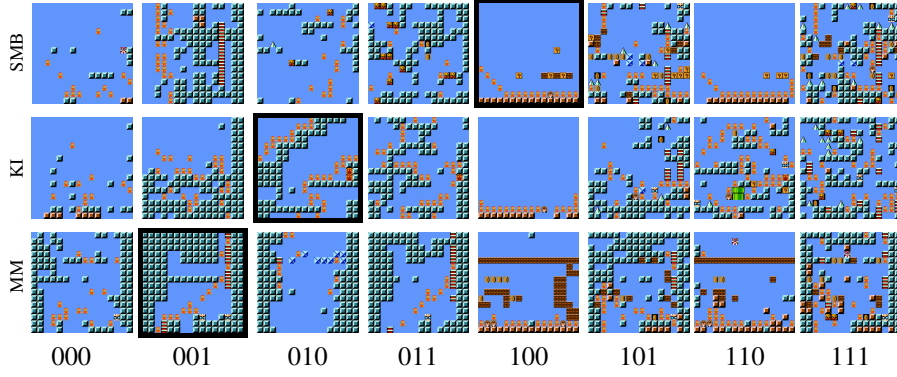
Fig. 6: Segments generated by conditioning on blend labels (shown below the 3rd row), using an original segment from *Super Mario Bros.* (SMB) (1st row), *Kid Icarus* (KI) (2nd) and *Mega Man* (MM) (3rd). First, second and third elements of the label correspond to SMB, KI and MM respectively. Bordered segments are originals. Reproduced with permission from [19]
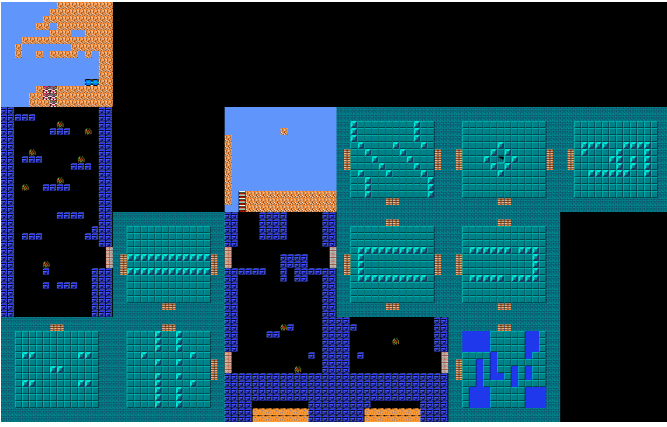

Fig. 7: Example level blending *The Legend of Zelda*, *Metroid* and *Mega Man*. Reproduced with permission from [20]

vectors. Example output levels are shown in Figure 2 and Figure 3 respectively. In follow-up work, Summerville et al. [56] extracted jump mechanics using paths in the generated blended levels. Example extracted jump arcs are shown in Figure 5. Here, the output knowledge additionally included the blended jump mechanics extracted from the blended levels.

In follow-up work, Sarkar et al. [19] and Sarkar and Cooper [20] used conditional VAEs which are VAEs that additionally use labels to control properties of generated, blended levels such as which combination of games to blend as well as orientation. In terms of the PCG-KT framework, an interesting distinction in these CVAE-based works is that in producing multiple blended domains as outputs, they exhibit a many-to-many cardinality as opposed to the many-to-one cardinality of the VAE-based works since the latter only produce a single blended domain as output. Example levels from [19] are shown in Figure 6. An additional point to note is that by enabling control over orientation, the approach in [20] enables producing levels combining dungeon rooms with platformer level segments (e.g. Figure 7). Thus this approach has an even higher content distance between input and transformed knowledge than prior methods which blend only platformers. Note that in these conditional approaches, the input and output

knowledge is similar to that of VAE-based methods, but here the transformation function of conceptual blending is performed via label manipulation.

2) Conceptual Expansion: Guzdial and Riedl introduced conceptual expansion as an alternative to traditional, non-latent conceptual blending meant to allow for the blending of machine learned knowledge from more than two sources [35]. The intuition is that it breaks the combinational creativity process into a series of functions of the form $CE^X(A, F) = \alpha_0 * f_0 + \alpha_1 * f_1...\alpha_n * f_n$. Here, F is a set of existing, machine-learned features, A is a set of filters that describe what to take from each feature and X is a goal. X may be an explicit piece of knowledge to recreate or a heuristic defining acceptability. Thus, by changing values of F and A an optimization approach can search a space of possible combinations of some given set of features. Optimizing these variables represents the transformation function in Conceptual Expansion work.

In [35], Guzdial and Riedl applied conceptual expansion to machine learned models of *Super Mario Bros.* level design to produce new types of output levels (e.g., combining boss levels and aboveground levels to create boss-aboveground levels). They also made use of conceptual blending and two other combinational creativity approaches as baselines. Notably, the levels were not being directly recombined, but instead the input knowledge was learned Bayesian network probabilities that represented level design models. After conceptual expansion was applied as the transformation function, the output was a novel Bayesian network that could produce unseen new level types as can be seen in Figure 8

In [7], Guzdial and Riedl applied Conceptual Expansion to machine learned representations of game rules and game levels for *Super Mario Bros.*, *Kirby's Adventure* [70], and *Mega Man*. This input knowledge took the form of knowledge graphs, combining the game rule and game level knowledge, both derived from gameplay video as the source of the raw knowledge. They found that conceptual expansion could outperform conceptual blending and genetic algorithms at recreating a held-out game given the other two games and a partial specification of the held-out game. However, in this work, Guzdial and Riedl did not attempt to produce
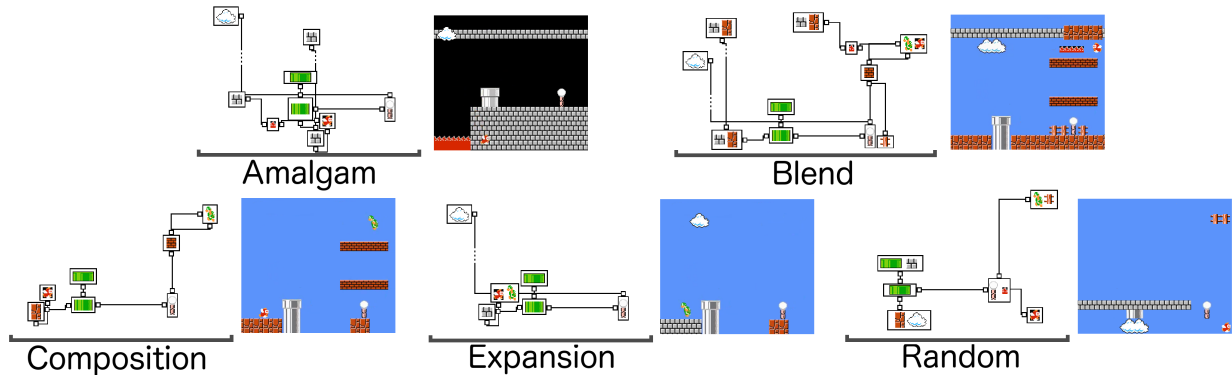
Fig. 8: Examples of the different final models and an output level for the different combinational creativity approaches when combining "castle" and "aboveground" *Super Mario Bros.* levels. Reproduced with permission from [35].
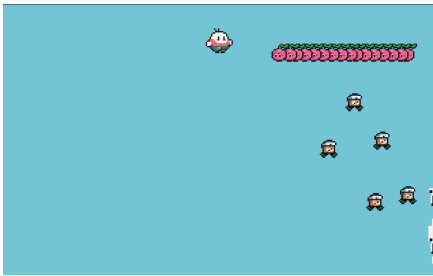

Fig. 9: A screenshot of the beginning of one of the Conceptual Expansion games. Reproduced with permission from [18].

fully novel games. Instead the output knowledge was simply an approximation of a held-out game based on a partial specification of that held-out game.

In [18], Guzdial and Riedl demonstrated the production of novel games from the same input knowledge: the graph of machine learned level and rule information from the three above games. They evaluated these games in a human subject study compared to other combinational creativity approaches. Their transformation function optimized output graphs using a hand-authored heuristic and greedy search over the conceptual expansion representation. Their results indicated that the conceptual expansion games outperformed games produced by existing combinational creativity methods, but that the heuristic was poorly designed for game quality. Determining how best to guide the search of possible combinations is still an important open problem. We include a screenshot of one of the games in Figure 9.

B. Transfer Learning

Transfer learning is a machine learning technique that focuses on transferring information or knowledge from one domain and applying it in another. In the following sections, we discuss prior PCG-KT methods that have made use of transfer learning as the transformation function.

1) Domain Adaptation: This refers to approaches for transforming knowledge extracted from a source domain to supplement or approximate the knowledge of another target domain. The existing domain adaptation approaches for level generation focus on finding a mapping between the representations of the source and target domains.

In [9], Snodgrass and Ontañón presented an approach aimed at automatically defining a mapping between the representations of multiple platforming game level domains. The core of this approach works by converting the source domain levels to the target domain representation using a subset of mappings between the domains, and filtering out the poorly performing mappings. The potential mappings are evaluated comparing frequency of tile occurrences and structures (e.g., 2×2 tile blocks) in the transformed input and target domains. The input knowledge for this approach is the set of tile levels from the input domain. The transformation function here is the mapping pipeline that tries to define how to map the tiles from the input domain to tiles in the target domain. After applying this mapping, the output knowledge is a set of levels re-represented as target domain levels. We include examples of different output levels for different mappings in Figure 10.

Unlike the above work which aimed to transform from a source domain to supplement a target domain, Snodgrass [36] and Snodgrass and Sarkar [14] aimed to transform a source domain to an intermediate target domain, and then either back to the source domain [36] or to a new secondary target domain [14]. In these approaches, the authors transform the source domain levels into an abstract domain representation meant to capture only the structural properties of the levels from each source domain (i.e., map each tile type to either a solid type or empty type). In [36], they use the abstract levels as input to a level generator that creates variations on that level. The variations are generated by matching the structures in the abstract level with structures in the source domain, and using regions of the source domain levels to fill in the details of the abstract level. In [14], they train VAEs on the abstract levels, and generate new levels in the abstract domain. They then use those generated abstract levels as input to a generator that blends the source domains instead of transforming back to an individual source domain. In this approach the blended levels are created by matching structures in the generated abstract levels with structures across all the source domains, and using the various source domains to fill in the details of the generated abstract level. Thus, the
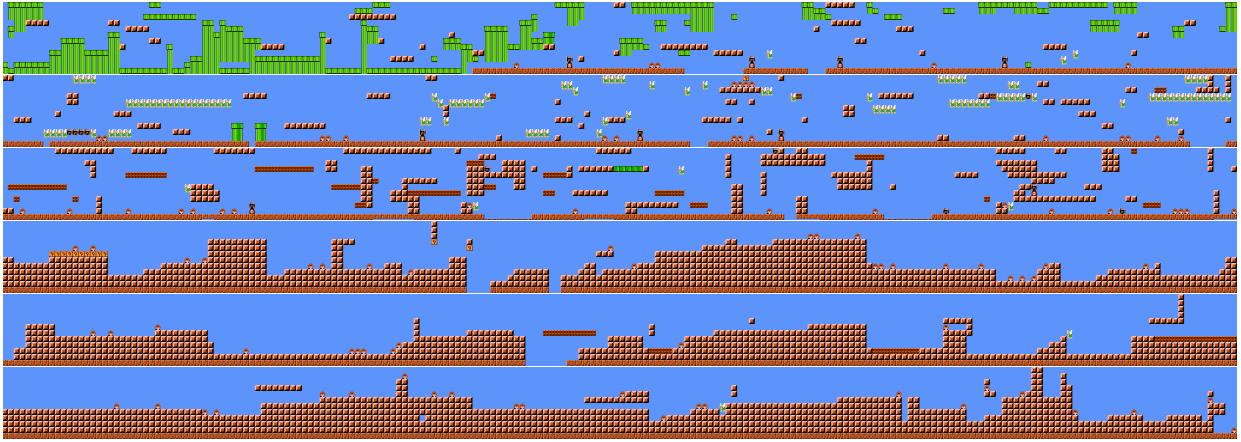
Fig. 10: Output levels from domain adaptation Markov models from *Super Mario Bros.* to *Kid Icarus* and *Kid Kool*. Reproduced with permission from [9].
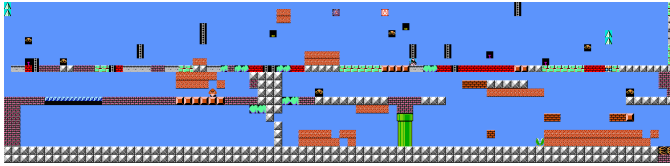

Fig. 11: Example *Castlevania* levels produced by the combined domain adaptation and conceptual blending approach. Reproduced with permission from [14].

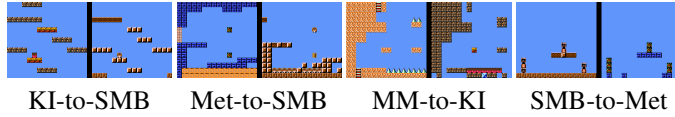KI-to-SMB   Met-to-SMB   MM-to-KI   SMB-to-Met

Fig. 12: Example style-transferred levels involving *Kid Icarus* (KI), *Super Mario Bros.* (SMB), *Mega Man* (MM) and *Metroid* (Met). Reproduced with permission from [37]

Fig. 13: Constructive (left) and search-based (right) generators built from the generalized 3x3s. Reproduced with permission from [38].

input knowledge here consists of the levels in the abstract representation, extracted from the original input levels and the output knowledge consists of the blended levels obtained via applying the transformation function which in this case is the structure matching process described above. Notice while [14] consists of both a domain adaptation stage and a combinational creativity stage, we include it in this section because the former is what enables the latter in this approach. Figure 11 contains example *Castlevania* levels produced with this approach.

2) Platformer Style Transfer: Style transfer [72] is a popular application of creative ML in visual art and has been the subject of a large body of research [73]. To implement style transfer for games, Sarkar and Cooper introduced tile2tile [37], a method for transferring style between levels of different platformer games. In the context of PCG-KT, we can view style transfer in games as extracting knowledge about the style of one game and transforming it into that of another game. While game blending is similar to style transfer, it is distinct. Style transfer translates the style of one domain to that of another where as in blending, the goal is to take the style and/or content of two domains and combine them to create
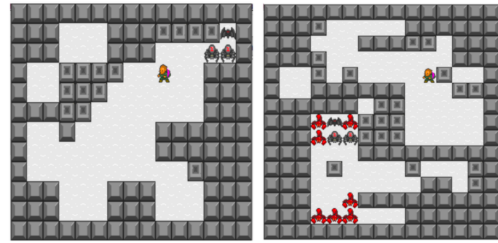
something new. The tile2tile approach involves translating levels from a source game into an affordance-based representation using a pre-defined mapping and then using a machine-learned model to convert from the affordance representation to the game-specific tiles of the target game. This approach treats the underlying affordances and the overarching game-specific tile distribution of a level as being respectively analogous to the concepts of content and style in visual art style transfer. The input knowledge here consists of the levels of the source game, the output knowledge consists of the same levels style-transferred to the target game and the transformation function is the combination of the hand-authored tile-to-affordance mapping and the model trained to translate from affordances to tiles of the target game. Example style-transferred levels are shown in Figure 12.

3) Design Pattern Approach for Multi-Game Level Generation: In [38], Beaupre et al. took a generalization and selection approach to procedurally generate new levels of particular games by generalizing across level design patterns from multiple games. This led to new game levels that did not fit within the set of existing levels, such as Pacman levels with collectibles outside of the default Pacman pellets. Their approach took all the game levels from the VGDL and re-represented each level in a more general representation (e.g., all enemies now considered identical, all collectibles now considered identical, etc.). From this representation, 3x3 patterns were extracted, which then informed constructive and search-
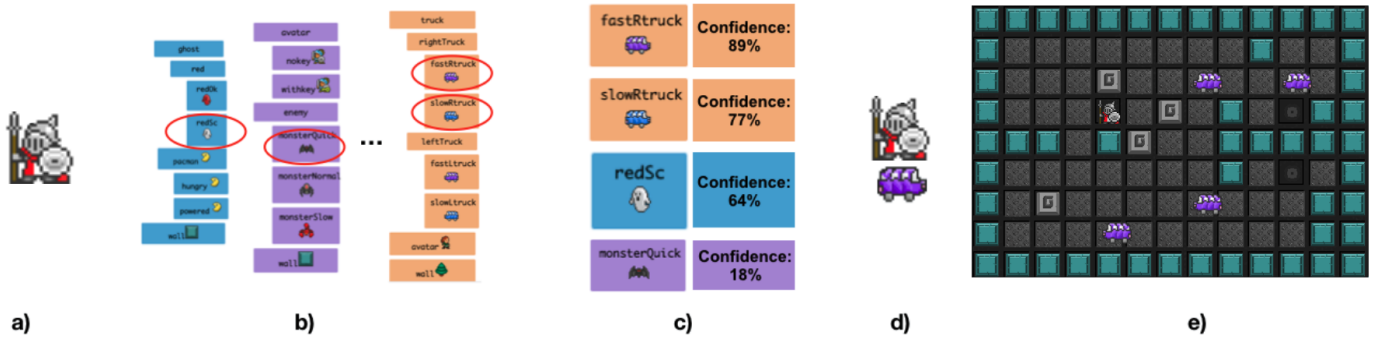
Fig. 14: Recommendation process for *Pitako*. Reproduced with permission from [71]. In this context, all sprites are associated with a game behavior e.g. shooter, chaser, random move, etc. In **a)**, users provide a sprite to the game engine. In **b)**, *Pitako* ranks the best sprite matches from other games (red circles) through data mining association (using confidence as a sorting criteria). In **c)**, users have access to the ranks. In **d)**, after inspecting the ranks, users select a new sprite. In **e)**, users can inspect the two sprites in action—the one they provided and the one suggested by the system, imported from another game.

based PCG generators as seen in Figure 13. Notably, this approach did not make use of statistical machine learning, but instead utilized an approach similar to case-based reasoning. In this work, the input knowledge consisted of the generalized 3x3 design patterns extracted from the original VGDL levels. The output knowledge was made up of the generated levels which in turn were built up using the extracted design patterns via an evolution-based transformation function.

4) Data mining association for game rule recombination: In [39], Machado et al. introduced Pitako, a recommender system for game mechanics for use within their prior game design tool Cicero. The system is built upon the VGDL and GVGAI (General Video Game AI) framework [74] and uses association rule mining to make suggestions to the designer regarding adding game elements and rules. Typically used with transaction databases, an association rule of the form $X \to Y$ says that the presence of an item X in a transaction implies the presence of item Y in the same transaction with some probability. To contextualize this in a game scenario, consider *Mega Man X* [75] and *Super Metroid* [76]. Both have a shooting mechanic and also share the "hold button for powerful shot" mechanic. That is, if a game implements a shooting mechanic, it may also implement the powerful shot rule. Framed as a transaction, having a shooting mechanic implies the presence of the powerful shooting mechanic in the same game. Association rules are discovered using data mining association algorithms such as the Apriori algorithm [77] to analyze databases of transactions. For this purpose, Pitako uses a game catalog. The system uses the games available in the GVGAI framework and applies a depth-first tree search on each of them to read all the game descriptions and put them in the catalog where every game element, from every game, can be disassociated from its original source and easily inserted in a target game. The catalog is organized to facilitate that its items, which are the elements required to describe VGDL games, can be categorized to be used as input for data mining association algorithms. Figure 14 outlines the Pitako recommendation process. It starts by getting the designer game description set. In (a), the user has only one game element. In (b), the system uses frequent itemset data

mining to identify associations that match the element in the designer's game to elements in the catalog of games. The red ellipses represent elements from different games which are candidates for being recommended. This procedure yields a list of recommendations sorted in descending order by the rule's confidence level shown in (c). The designer picks the recommendation with the highest confidence level (d) and employs it as a new element in the game. The element shown here is from a Frogger clone and in this example, the user is mixing elements of Frogger and Sokoban. When the user picks a recommendation, its game elements are transferred from a game in the catalog and inserted into the in-development game. Thus, the input knowledge here is the game catalog extracted from the GVGAI games using search while the output knowledge is the designed game which contains elements taken from the input games that the user chose. The transformation process involves both the application of the Apriori algorithm on the catalog to generate recommendations as well as the use of these recommendations by users when designing their game. It is interesting to note that the transformation process here is mixed-initiative as opposed to the more automated transformations in other surveyed work.

VI. DISCUSSION

In this section, we summarize open problems and directions for future work related to PCG-KT.

A. Evaluation

It is unclear how to best evaluate knowledge transformation in a generative process. Existing works have utilized techniques such as expressive range analysis [78], playability measures, and user studies, while ML-based systems have used comparisons between training and generated distributions [79]. However, these focus on assessing the goodness and functionality of generated outputs rather than evaluating the properties of the transformation itself. Moreover, in certain cases, even content evaluation methods need to be improved upon. For example, in the case of game and level blending, it remains unclear how best to assess the quality of blended

levels and/or levels belonging to an entirely new domain produced by blending. A current common approach is to compare content produced in the new/blended domain with content belonging to the original input domains, however these often signal whether blending or transformation has taken place rather than provide information about the extent, type and quality of blending and transformation. This is analogous to our concept of measuring the content distance of the transformation function introduced above, but tells us little about the other properties of the transformation. Thus in the future, more robust and informative evaluation techniques need to be developed. This would likely require defining specific metrics capable of assessing various properties of knowledge transformation, perhaps utilizing some of the other features highlighted in our framework definition.

B. Extending to Multiple Game Genres

A majority of works utilizing knowledge transformation (and PCG in general) have focused on platformer games. While using games of one genre is more convenient, restricting to one genre limits the scope and possibility space of the transformation process. Extending knowledge transformation approaches to multiple genres and domains could enable the generation of new types and varieties of content. Understandably, working with knowledge extracted from multiple genres would pose new challenges in terms of developing techniques for reasoning with such knowledge and applying transformations in a meaningful way. However, being able to do so could promise novel gameplay experiences. Could we design transformation functions able to combine extracted knowledge from a platformer such as Mario with extracted knowledge from an adventure game like Zelda? As previously discussed, Sarkar and Cooper [20] took a step in this direction by generating levels that combined platformer segments with dungeon rooms but ignored considerations of mechanics in such levels. Addressing such issues and extracting knowledge from different genres and domains in a manner that is conducive to meaningful combinations and transformations may open up the possibility of building systems that could blend genres of games rather than just levels or games. For example, could such a system combine the mechanics of a platformer with the lock-and-key progression of an adventure game to produce a metroidvania? In terms of the PCG-KT framework, such a system would exhibit transformation properties of even higher content distance between input and output knowledge since the output domain of the transformation function would not only be a new game but possibly a new genre altogether.

C. Combining Models and Techniques

In all previously discussed works, the knowledge transformation process was performed using one primary model or architecture such as Markov chains, Bayes Nets and VAEs. In the future, it would be interesting to consider hybrid knowledge transformation processes that combine the use of these models in some meaningful way. The choice of model used as the derivation function dictates the form of the extracted knowledge. We have seen this take the form of latent vector representations in the case of VAEs and probability tables in the case of Markov chains, when applied to similar raw knowledge. Thus, it might be fruitful to explore how these different extracted knowledge representations could be combined to inform and influence the transformation process and obtain different varieties of output knowledge. For example, could the models used for conceptual expansion be combined with the Markov models used for domain adaptation or with VAEs to learn a continuous latent space of game graphs? Alternatively, could conceptual expansion be applied on latent vectors of different games or the design patterns of some other prior works? Such model ensembles could be viewed as hierarchical knowledge transformation where the knowledge extracted from one derivation function is used as the input to another. In other words, this can be seen as replacing the single derivation function D with a composition of derivation functions $D_n \circ D_m \circ ... \circ D_b \circ D_a$, whose application produces intermediate extracted knowledge $\mathbb{K}_a, \mathbb{K}_b...\mathbb{K}_m, \mathbb{K}_n$ at each step. Examining the features of such successively extracted knowledge and how they might influence the transformed knowledge could be another fertile direction for future work. Similarly, one could conceive of a composition of transformation functions $T_n \circ T_m \circ ... \circ T_a$ that performs successive transformations on some given extracted knowledge, producing new transformed knowledge at each step. This could take the form of a system that produces game blends (via conceptual blending) and then transfers the style of the blended game to another game (via domain transfer).

D. Design Tools and Controllability

An important direction for future work is to add more controllability to knowledge transformation processes as well as to democratize PCG-KT models via user-facing designer tools. For example, consider the blending related works discussed earlier. While these enable controllability via specifying objective functions and supervised ML models, they require experience and familiarity with ML and evolutionary algorithms, thereby making it feasible for only researchers and experts to work with such methods. Additionally, several methods discussed in this paper, particularly those based on ML, can be viewed as black boxes that take in input (and optionally a set of constraints) and produce an output without affording much room for user interaction with the generative/transformation process, thereby allowing little authorial control. One could imagine future design tools addressing such concerns by, for example, visualizing the latent design spaces learned by these models and allowing users to interactively navigate such spaces and steer the generation process. Overall, future work needs to focus on building tools that enable a wider variety of users to control such models (e.g., enabling the use of UI elements and interfaces rather than defining one-hot encoded labels and coding up fitness functions for evolution). Such tools could take inspiration from the large body of existing co-creative systems [80], particularly those based on ML, such as Morai Maker [81] and TOAD-GUI [82], both for making Mario levels, the tool from Schrum et al. [83] for interactively exploring the latent spaces of GANs trained

on Mario and Zelda, as well as Lode Encoder [84], a VAE-based tool for generating *Lode Runner* [85] levels. Note that in all these examples, the tools only work in one game/domain and thus cannot be considered as tools for PCG-KT. In fact, the Cicero and Pitako tools discussed previously are among the few examples of PCG-KT design tools, thus speaking to the need for developing more of these in the future. Sarkar and Cooper [86] provide a blueprint for such a tool but at the time of writing, no implementation exists. Incorporating more controllability into PCG-KT approaches and operationalizing them via user-friendly tools could further advance the field and help PCG-KT methods be applied for game and level design more widely. In addition to tools that help less experienced users, future works could also consider tools that incorporate user knowledge in performing the transformation process. These could be viewed as creativity support tools [87] where the user enters input knowledge taken from various domains and the tool enables the user to either choose from a variety of automated derivation functions to apply, create their own hand-authored derivation functions or highlight aspects of the input knowledge that are amenable to transformation. Such tools could thus help incorporate a user's creative insights into the PCG-KT process and foster a more co-creative, human-centered form of PCG-KT as opposed to the more automated methods that we have surveyed in the paper.

VII. Conclusion

In this paper, we introduced Procedural Content Generation via Knowledge Transformation (PCG-KT), a new lens for PCG research that focuses on methods that generate content via transforming knowledge between domains. We offered a formal definition for PCG-KT, outlined a framework for characterizing works under this lens, discussed several examples of existing works that fall under the framework and concluded with directions for future work. Prior survey papers on search-based PCG [1] and PCGML [5] have been successful in highlighting a specific family of PCG approaches as well as pointing out future work directions which inspired further research. For example, several works highlighted in this paper were directly influenced by suggestions made in the PCGML paper. We hope to achieve a similar result with this paper. By highlighting this new branch of PCG, we wish to stimulate further research in this interesting and exciting new direction.

References

[1] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation," in *Proceedings of the European Conference on the Applications of Evolutionary Computation*. Springer, 2010, pp. 141–150.

[2] S. Risi and J. Togelius, "Increasing generality in machine learning through procedural content generation," *Nature Machine Intelligence*, vol. 2, no. 8, pp. 428–436, 2020.

[3] G. Smith, "What do we value in procedural content generation?" in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, 2017, pp. 1–2.

[4] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, 2011.

[5] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (PCGML)," *IEEE Transactions on Games*, vol. 10, no. 3, pp. 257–270, 2018.

[6] M. Cook, S. Colton, and J. Gow, "The angelina videogame design system—part i," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 2, pp. 192–203, 2016.

[7] M. Guzdial and M. Riedl, "Automated game design via conceptual expansion," in *Proceedings of the 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2018.

[8] M. Guzdial, N. Liao, J. Chen, S.-Y. Chen, S. Shah, V. Shah, J. Reno, G. Smith, and M. O. Riedl, "Friend, collaborator, student, manager: How design of an AI-driven game level editor affects creators," in *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–13.

[9] S. Snodgrass and S. Ontañón, "An approach to domain transfer in procedural content generation of two-dimensional videogame levels," in *Proceedings of the 12th Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[10] A. J. Summerville, S. Snodgrass, M. Mateas, and S. O. Villar, "The VGLC: The video game level corpus," *Proceedings of the 7th Workshop on Procedural Content Generation*, 2016.

[11] S. Snodgrass and S. Ontañón, "Learning to generate video game maps using Markov models," *IEEE transactions on computational intelligence and AI in games*, vol. 9, no. 4, pp. 410–422, 2016.

[12] A. Summerville and M. Mateas, "Super Mario as a string: Platformer level generation via LSTMs," in *Proceedings of the 1st International Joint Conference of DiGRA and FDG*, 2016.

[13] M. Guzdial and M. Riedl, "Learning to blend computer game levels," in *Proceedings of the 7th International Conference on Computational Creativity*, 2016.

[14] S. Snodgrass and A. Sarkar, "Multi-domain level generation and blending with sketches via example-driven BSP and variational autoencoders," in *Proceedings of the 15th International Conference on the Foundations of Digital Games*, 2020.

[15] S. Snodgrass and S. Ontañón, "A hierarchical MdMC approach to 2d video game map generation," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 11, no. 1, 2015.

[16] A. Sarkar, Z. Yang, and S. Cooper, "Controllable level blending between games using variational autoencoders," in *Experimental AI in Games Workshop at AIIDE*, 2019.

[17] A. Sarkar, A. Summerville, S. Snodgrass, G. Bentley, and J. Osborn, "Exploring level blending across platformers via paths and affordances," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2020.

[18] M. Guzdial and M. O. Riedl, "Conceptual game expansion," *IEEE Transactions on Games*, vol. 14, no. 1, pp. 93–106, 2021.

[19] A. Sarkar, Z. Yang, and S. Cooper, "Conditional level generation and game blending," in *Experimental AI in Games Workshop at AIIDE*, 2020.

[20] A. Sarkar and S. Cooper, "Dungeon and platformer level blending and generation using conditional vaes," in *Proceedings of the IEEE Conference on Games (CoG)*, 2021.

[21] J. Permar and B. Magerko, "A conceptual blending approach to the generation of cognitive scripts for interactive narrative," in *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.

[22] S. Snodgrass and S. Ontañón, "Experiments in map generation using Markov chains." in *Proceedings of the International Conference on the Foundations of Digital Games*, 2014.

[23] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-oriented software architecture: a system of patterns*. John Wiley & Sons, 2008, vol. 1.

[24] C. Alexander, *A pattern language: towns, buildings, construction*. Oxford university press, 1977.

[25] S. Björk, S. Lundgren, and J. Holopainen, "Game design patterns," in *Proceedings of Digital Games Research Conference*, 2003.

[26] B. Kreimeier. (2002, March) The case for game design patterns. [Online]. Available: https://www.gamedeveloper.com/design/the-case-for-game-design-patterns

[27] S. Dahlskog and J. Togelius, "Patterns and procedural content generation: Revisiting mario in world 1 level 1," in *Proceedings of the 1st Workshop on Design Patterns in Games*, 2012, pp. 1–8.

[28] K. Hullett and J. Whitehead, "Design patterns in FPS levels," in *Proceedings of the 5th International Conference on the Foundations of Digital Games*. Association for Computing Machinery, 2010, p. 78–85.

[29] A. Khalifa, F. de Mesentier Silva, and J. Togelius, "Level design patterns in 2d games," in *Proceedings of the IEEE Conference on Games (CoG)*, 2019.

[30] B. Mears and J. Zhu, "Design patterns for silent player characters in narrative-driven games," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, 2017.

[31] K. Hullett, "Cause-effect relationships between design patterns and designer intent in FPS levels," in *Proceedings of the 2nd Workshop on Design Patterns in Games (DPG)*, vol. 13, 2013.

[32] A. Summerville, J. Osborn, C. Holmgård, and D. W. Zhang, "Mechanics automatically recognized via interactive observation: Jumping," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, 2017, pp. 1–10.

[33] M. J. Nelson and M. Mateas, "Towards automated game design," in *Congress of the Italian Association for Artificial Intelligence*. Springer, 2007, pp. 626–637.

[34] J. Gow and J. Corneli, "Towards generating novel games using conceptual blending," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2015.

[35] M. J. Guzdial and M. O. Riedl, "Combinatorial creativity for procedural content generation via machine learning," in *Workshops at the 32nd AAAI Conference on Artificial Intelligence*, 2018.

[36] S. Snodgrass, "Levels from sketches with example-driven binary space partition," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 15, no. 1, 2019, pp. 73–79.

[37] A. Sarkar and S. Cooper, "tile2tile: Learning game filters for platformer style transfer," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2022.

[38] S. Beaupre, T. Wiles, S. Briggs, and G. Smith, "A design pattern approach for multi-game level generation," in *Proceedings of the 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2018.

[39] T. Machado, D. Gopstein, A. Nealen, and J. Togelius, "Pitako-recommending game design elements in cicero," in *Proceedings of the IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.

[40] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[41] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2010.

[42] D. S. Chaplot, G. Lample, K. M. Sathyendra, and R. Salakhutdinov, "Transfer deep reinforcement learning in 3d environments: An empirical study," in *NeurIPS Deep Reinforcement Learning Workshop*, 2016.

[43] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[44] K. Kansky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George, "Schema networks: Zero-shot transfer with a generative causal model of intuitive physics," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1809–1818.

[45] A. Melnik, S. Fleer, M. Schilling, and H. Ritter, "Functional learning-modules for atari ball games," in *ICLR Workshop Track*, 2018.

[46] A. Mittel and P. S. Munukutla, "Visual transfer between atari games using competitive reinforcement learning," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019, pp. 499–501.

[47] S. Gamrian and Y. Goldberg, "Transfer learning for related reinforcement learning tasks via image-to-image translation," in *International conference on machine learning*. PMLR, 2019, pp. 2063–2072.

[48] S. W. Kim, Y. Zhou, J. Philion, A. Torralba, and S. Fidler, "Learning to simulate dynamic environments with gameGAN," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1231–1240.

[49] S. Colton and G. A. Wiggins, "Computational creativity: The final frontier?" in *Proceedings of the European Conference on Artificial Intelligence*. IOS Press, 2012, pp. 21–26.

[50] A. Liapis, G. N. Yannakakis, and J. Togelius, "Computational game creativity," in *Proceedings of the International Conference on Computational Creativity*, 2014.

[51] M. A. Boden, "Creativity," in *Artificial intelligence*. Elsevier, 1996, pp. 267–291.

[52] G. Fauconnier and M. Turner, "Conceptual integration networks," *Cognitive Science*, vol. 22, no. 2, pp. 133–187, 1998. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1207/s15516709cog2202_1/pdf

[53] C. Gagné and E. Shoben, "Influence of thematic relations on the comprehension of modifier–noun combinations," *Journal of Experimental Psychology: Learning, Memory and Cognition*, vol. 23, pp. 71–87, 1997.

[54] J. Lehman and K. O. Stanley, "Novelty search and the problem with objectives," in *Genetic programming theory and practice IX*. Springer, 2011, pp. 37–56.

[55] D. Gravina, A. Liapis, and G. Yannakakis, "Surprise search: Beyond objectives and novelty," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2016, pp. 677–684.

[56] A. Summerville, A. Sarkar, S. Snodgrass, and J. Osborn, "Extracting physics from blended platformer game levels," in *Experimental AI in Games Workshop at AIIDE*, 2020.

[57] M. A. Boden, "Computer models of creativity," *AI Magazine*, vol. 30, no. 3, pp. 23–23, 2009.

[58] S. Ontanón and E. Plaza, "Amalgams: A formal approach for combining multiple case solutions," in *Proceedings of the International Conference on Case-Based Reasoning*. Springer, 2010, pp. 257–271.

[59] K. Badie and M. T. Mahmoudi, "Compositional adaptation in case-based reasoning based on the semantic relations between the components in the cases," in *2017 IEEE International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, 2017, pp. 449–454.

[60] T. Schaul, "An extensible description language for video games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 325–331, 2014.

[61] Konami, "*Frogger*," Game [Arcade], 1981.

[62] Nintendo, "*The Legend of Zelda*," Game [NES], 1986.

[63] D. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2013.

[64] Nintendo, "*Super Mario Bros.*" Game [NES], 1985.

[65] ——, "*Kid Icarus*," Game [NES], 1986.

[66] ——, "*Metroid*," Game [NES], 1986.

[67] Capcom, "*Mega Man*," Game [NES], 1987.

[68] Konami, "*Castlevania*," Game, 1986.

[69] Tecmo, "*Ninja Gaiden*," Game, 1988.

[70] H. Laboratory, "*Kirby's Adventure*," Game [NES], 1993.

[71] T. Machado, D. Gopstein, A. Wang, O. Nov, A. Nealen, and J. Togelius, "Evaluation of a recommender system for assisting novice game designers," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 15, no. 1, 2019, pp. 167–173.

[72] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv preprint arXiv:1508.06576*, 2015.

[73] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song, "Neural style transfer: A review," *IEEE transactions on visualization and computer graphics*, 2019.

[74] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms," *IEEE Transactions on Games*, vol. 11, no. 3, pp. 195–214, 2019.

[75] Capcom, "*Mega Man X*," Game [SNES], 1993.

[76] Nintendo, "*Super Metroid*," Game [SNES], 1994.

[77] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, vol. 1215, 1994, pp. 487–499.

[78] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 2010.

[79] A. Summerville, "Expanding Expressive Range: Evaluation Methodologies for Procedural Content Generation," in *Proceedings of the 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2018.

[80] G. N. Yannakakis, A. Liapis, and C. Alexopoulos, "Mixed-initiative co-creativity," in *Proceedings of the International Conference on the Foundations of Digital Games*, 2014.

[81] M. Guzdial, N. Liao, and M. Riedl, "Co-creative level design via machine learning," *Experimental AI in Games Workshop at AIIDE*, 2018.

[82] F. Schubert, M. Awiszus, and B. Rosenhahn, "Toad-GAN: a flexible framework for few-shot level generation in token-based games," *IEEE Transactions on Games*, 2021.

[83] J. Schrum, J. Gutierrez, V. Volz, J. Liu, S. Lucas, and S. Risi, "Interactive evolution and exploration within latent level-design space of generative adversarial networks," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2020, pp. 148–156.

[84] D. Bhaumik, A. Khalifa, and J. Togelius, "Lode encoder: AI-constrained co-creativity," in *Proceedings of the IEEE Conference on Games (CoG)*, 2021.

[85] D. Smith, "*Lode Runner*," Game, 1983.

[86] A. Sarkar and S. Cooper, "Towards game design via creative machine learning (GDCML)," in *Proceedings of the IEEE Conference on Games (CoG)*, 2020.

[87] B. Shneiderman, "Creativity support tools: accelerating discovery and innovation," *Communications of the ACM*, vol. 50, no. 12, pp. 20–32, 2007.