Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2025)

Game Design Is Generative Design

Michael Cook

Human-Centred Computing Group
King's College London
mike@possibilityspace.org

Abstract

Generative design – the use of randomness, simulation or complex algorithms as part of a creative work – remains an uncommon skillset in the games industry. Acquiring familiarity and confidence with generative concepts is difficult and intimidating, which hampers its uptake in games. In this paper we argue that game design *is* generative design, and that by embracing the two as one and the same we can find new ways to communicate generative thinking to those who are unfamiliar with it. We discuss the history of procedural generation in research and design, as well as reporting on surveys of 261 players and 126 game designers; we introduce the term *procedural gameplay system* to define a subset of generative systems used in games; and we offer some insights into how drawing an equivalence between these two design practices can yield new approaches and perspectives.

Introduction

Procedural content generation (PCG) has been in use in videogames for at least forty years, but in the last fifteen we have seen a dramatic increase in how popular it is with players and how commonly it is applied to games. Three of the five best-selling PC games of all time use procedural generation at their core, and all were released within twelve months of one another at the beginning of the 2010s (*Spelunky*, *Minecraft* and *The Binding of Isaac*).

However, it is not quite clear how PCG fits into the scope of the games industry or the many subfields that make up game development. Research into procedural generation, for example, is frequently published in artificial intelligence conferences, and new techniques are published in books alongside writing about AI engineering (Brewer 2021). We searched the industry-focused job listing site Game Jobs Direct for listings with the word 'procedural' in the description in June 2024. Of the forty listings found, only five included game design as *part* of the role, and all five were *technical* designer roles with an emphasis on 'data [and] software architecture'. Most of the results were in art and animation, or in systems engineering.

Yet there clearly is a healthy overlap between procedural generation and game design. In the edited collection *Procedural Generation in Game Design*, twenty of the twenty-four chapter authors have credits as designers on commercial games, or design games independently. Searching GDC talks between 2020-2022[1] for the term 'procedural' yields 32 talks spread across 13 tracks, including 7 each in Visual Arts and Programming, 4 each in Design and Animation, 3 in AI and 2 in Audio. Clearly, procedural generation is relevant to almost every aspect of game production.

Despite the apparent widespread use in industry, procedural generation's development as a creative discipline, area of study and technical field remains stunted. In its application within games, for example, we see it applied in mostly the same ways it has been used for decades. Outside of its use by artists and animators, certain types of game content, usually levels and items, are most commonly generated, with template-based grammars and 2D noise being the most common techniques used (which we support later with our survey of players). Outliers exist, and some of the most celebrated applications break these trends, but they remain the exception (such as *No Man's Sky*).

In the broader games community the attitude towards generative systems retains a healthy amount of skepticism and negativity. Even before the advent of 'generative AI' systems such as ChatGPT, many game developers avoided the use of procedural generation or similar techniques, stating reasons such as a belief it was 'too random' or too hard to control (Duncan 2015). Some games have even explicitly marketed themselves as containing 'hand-crafted' content (Metanet Software 2015), in opposition to the scale-based marketing of games such as Borderlands (Burke 2019).

Researchers have attempted to solve some of these problems and make procedural generation more accessible by developing tools and analytical techniques to support the use and understanding of procedural generation. Researchers have developed a range of techniques which showed promise, particularly *expressive range analysis* (Smith and Whitehead 2010), yet as we show in this paper their uptake within the games industry is limited and comparative studies of both procedural generators and evaluatory frameworks are rare. New ways of understanding generative systems are valuable to both researchers and developers, as they both support the dissemination of new ideas, and encourage the uptake of existing ones. Yet bridging the

[1] Talks after these dates were paywalled.

gap remains tricky.

Procedural generation has clearly been a driver for commercial success in the industry and led to many entertaining and impactful creative works; yet it is still misunderstood by both developers and players, and this gulf in understanding worsens as generative AI begins to spread through creative workflows. We believe that some of these problems stem from the way we have come to discuss and think about procedural content generation in games. The industry's understanding of the term has crystallised around a few specific use-cases, which has restricted innovation, amplified the perception of its shortcomings, and pigeonholed its use-cases to a handful of repetitive and commercially-focused applications. Without breaking this mindset, procedural generation cannot be fully embraced by the games industry, and is doomed to remain a niche technique for 'increasing replay value' and 'reducing the cost of development'.

In this paper we argue very simply that *all* game design is generative design. The distinction between 'procedural content' and the rest of a game experience is nominal: to design games is to design unpredictable, dynamic, ever-shifting experiences, and all game designers, bar none, think about the generative ramifications of their designs when they work. We can therefore reframe 'generative design' as a much broader way of thinking about experience design, and help break down the barriers between what we currently call 'procedural generation' and other forms of design work.

In doing so, we achieve two important things. First, we provide ways for game designers to view procedural content generation in a different light. By thinking about generative design as involving the same instincts, thought processes and skills as game design, we encourage designers to embrace this technique and develop it further as an artform, hopefully breaking down some of the barriers and skepticism that still surrounds this technique. Secondly, by drawing this connection we enable the transfer of theory and practice between the two domains. By thinking about procedural design as game design, and vice versa, we enable new ways of analysing, evaluating and building such systems.

The remainder of this paper is structured as follows: in *Background* we discuss the history of procedural generation research and theory, and redefine a subset of the field. In *Designing for Nondeterminism* we walk through a set of linked examples and consider structural comparisons between procedural generation and game design. In *Discussion* we link the argument in the paper to future work bringing game design and procedural generation research closer together. In *Conclusions* we provide concluding remarks.

Background

In this section we aim to establish three things: that definitions of procedural content generation are not in alignment and disagree on some key properties; that randomness is an integral part of how game designers use procedural generation; and that there is a mismatch between the popular understanding of a game which colloquially 'uses procedural generation' and a game which employs techniques that originated in the study of generative systems. In the latter case, we introduce a new term – procedural gameplay system – to define a subset of generative systems that players and designers treat differently.

Defining Procedural Generation

Procedural content generation does not have a commonly accepted definition either within research or the industry, however certain definitions in papers are widely-cited. One such example appears in (Togelius et al. 2013), where Togelius et al. define procedural content generation as 'the algorithmic generation of game content with limited or no human contribution'. This specific definition has either been explicitly cited or quoted wholesale in many subsequent publications in different areas of games research (Connor, Greig, and Kruse 2017) (Shaker et al. 2013) (Zafar, Mujtaba, and Beg 2020), and many other definitions inherit from it, including the PCG Book which is widely used as an educational resource.

Although this definition rings true intuitively, one notable weakness of it is that 'algorithmic' and 'content' are very loosely defined. Algorithmic could refer to any kind of process expressed in code, and the word 'procedural' simply means described by a formal procedure, a property which describes all code. 'Content' could refer to almost anything in a game, and this word is sometimes omitted from the term PCG altogether. Therefore, a procedure for loading saved games could be considered a procedural generator under such a definition – a process which involves no human contribution. If this seems far-fetched, consider that Elite's galaxy generation is often cited as an early example of procedural generation, and its primary purpose was to 'load' galaxies from their compressed form as a random seed – the game only includes a handful of galaxies to choose from (Frontier Developments 1984).

An important feature present in most procedural generators, but omitted from Togelius' definition, is that the content generated cannot be predicted or anticipated by the player in advance. In (Smith 2015) Smith observes that 'randomness is a hallmark of procedural content generation', although they partition their writing as a subset of Togelius' definition that is non-deterministic. Similarly, Doull defines procedural content as 'a random or pseudo-random process that results in an unpredictable range of game play spaces'(Doull 2008). We believe that randomness is not included in some definitions of procedural generation for two reasons: first, as Smith observes, the confounding presence of examples such as Elite, where unpredictability is not technically a property of the system in its final form within the game; and second, because content generation is often thought of as a stand-in for something that would otherwise have been hand-designed. Therefore the nature of its unpredictability is not considered a fundamental part of the system itself, rather a side effect of the way the hand-designed content is replaced. This might be due in part to the fact that procedural generation is often framed as a technical problem, tackled by engineers and programmers, or because of a need to motivate the technique as a tool for saving time or money, rather than achieving a design goal. Its success or failure is based on its ability to replicate people designing static content – the role of the system within the overall game's design

is less often considered, and therefore its unpredictability is not considered an inherently important feature. Karth and Compton address this in their paper *Procedural Generation is Impossible*:

> The terms of the "content arms race" imply that PCG is about replacing human-created art assets with machine-generated artifacts that are indistinguishable from what the humans would have created. (Karth and Compton 2023)

This assertion is uncited in the paper, but the authors do reference Will Wright's 2006 GDC talk in which he proposes the future of game design as being driven by content generation, both by procedures and users (Gouskos 2006). Wright approached this from the perspective of games production, and procedural content generation has been useful in this regard. Tools such as SpeedTree and Houdini have become very important to high-end AAA game development. However, this is not what players, critics and often even developers mean when they refer to procedural content generation. This mixing of perceptions of procedural content, where its application as an automation of labour is mixed or overlaps with its application as an expressive design tool, might contribute to the conflicted relationship the technique has with game developers.

To dig further into the popular understanding of this term, we conducted a survey of 261 people who had some experience of playing videogames, recruited via advertising on social media and Discord games communities[2]. 98% of participants said they play games every day or every week, and 95% of participants had played for over fifteen years. We presented them with several videogames and asked them first if they thought each game should be tagged with 'procedural generation' on a storefront like Steam, and second if they thought the game *used* procedural generation (independent on whether it should be tagged). A subset of results can be seen in Table 1. Games which traditionally use procedural generation to provide variable, changing and unpredictable game content, such as Spelunky and Minecraft, are overwhelmingly identified as games which employ procedural generation and should be tagged as such.

By contrast, Fortnite and Baldur's Gate 3 both had a majority of respondents indicating the the game does not use procedural generation and should not be tagged as such either. In both cases, however, there is a markedly higher proportion of responses indicating they don't know. Simply by perusing the credits of Baldur's Gate 3, we can see that they licensed SpeedTree for the game, one of the most commonly-cited examples of developer-side procedural content generation (Moby Games 2020). In 2022 Fortnite's development team published an article about their use of procedural music in the game (Block, Oakley, and Swanepoel 2022). These are just isolated examples of procedural systems used in these games yet, intuitively, they do not strike the player as being games which 'use procedural generation'.

In general, procedural content generation follows an 'I know it when I see it' definition, similar to concepts such as art, creativity or games themselves. However, our survey responses also suggest a perceptual split between a game being labelled as 'procedural generation' and a game *actually using* procedural generation techniques. There were three times as many people unsure about whether Fortnite 'uses' procedural generation than people unsure whether it should be *labelled* as a procedural game. No-one said that Baldur's Gate 3 should be labelled with 'procedural generation', but nearly 10% of respondents said they thought it used the technique. The gap between these concepts is notable and worth investigating.

Randomness in PCG and Game Design

In earlier decades, particularly the 2000s, procedural generation was often referred to as 'random generation'. Google Trends results suggest that this has fallen off in popularity in the last decade, likely because several popular games used the 'procedural' terminology around 2010. In his 2006 GDC talk, two years before the original release of Spelunky, Will Wright reportedly asked the audience what term he should use to refer to procedural content as his team was tired of hearing the phrase 'procedural', suggesting it was already catching on then (Gouskos 2006). Generative systems designers often object to the characterisation of their work as 'random', which may also have led to pushback against the phrase (McKendrick 2014). Some even attempt to distinguish the two as separate approaches (Bycer 2015).

Randomness implies, among other things, unpredictability. Yet the original use of the phrase 'random generation' possibly also implied a lack of design, a lack of intent, possibly even a lack of care, which may be why it was rejected as a term by practitioners (and researchers). The idea that something can be random and unpredictable, yet at the same time also be shaped, directed or channelled to a particular end, is everywhere around us in game design. However it is not an intuitively accepted notion by the general public, or even by most game designers or researchers. When we are asked to roll a twenty-sided die in a roleplaying game instead of two ten-sided dice, it is because an intentional design decision has been made to give us a particular random distribution of numbers. The behaviour of the dice is random, but their inclusion in the design is not.

In addition to our survey of players mentioned earlier, we also conducted a survey of 127 game developers, advertised on private developer mailing lists, communities and Discords, as well as social media. 63% of respondents indicated game development was their primary source of income, while 82.5% of respondents identified themselves as independent developers. 40% of respondents worked in teams of less than 20 people, while 41% were solo developers. In both developer and player surveys we asked respondents to write down the three words they most strongly associate with procedural generation. Table 2 shows the eight most common words for both groups – 'random' is the most common in both sets, and for players it is almost twice as common as the second most frequent response (not including 'randomness' which is also in both lists). This suggests that some aspect of the word 'random' – which is itself a multifaceted term – is important to the modern conception

[2]King's College London ethics ref. MRA-24/25-47575

Game	Should be tagged	Should not be tagged	Don't know	Uses	Does Not Use	Don't know
Minecraft	92.7%	2.7%	0.7%	93.1%	1.1%	2.3%
Spelunky	68.7%	0%	3.1%	69.2%	1.1%	5.4%
Fortnite	3.5%	62.3%	3.8%	13%	43.5%	12%
Baldur's Gate 3	0%	67.4%	7.8%	9.3%	46.9%	17.8%

Table 1: Survey responses about whether a game should have the tag 'procedural generation' on storefronts, and whether a game uses procedural generation. Remaining respondents indicated they did not know the game well enough to comment.

Word	Frequency (261 Players)	Word	Frequency (126 Developers)
random	69	random	27
roguelike	36	roguelike	20
randomness	16	infinite	11
infinite	16	variety	11
algorithm	14	replayability	11
replayability	14	content	11
repetitive	12	randomness	7
content	12	noise	7

Table 2: Responses to the question "What are three words you most strongly associate with 'procedural generation'?". For each group of respondents (players and game developers), we show the eight most common words in the response set.

of procedural generation.

Randomness is also an important part of general game design. *Input* and *output* randomness are two terms used to describe the different ways randomness can impact player decision-making in games, outside of the realm of content generation. The notion of input versus output randomness predates its discussion within videogames, but appears to have been popularised by Keith Burgun, who defines the difference between the two concepts as follows:

> Output randomness is noise injected between the player's decision and the outcome... [Input] randomness informs the player before [they] make [their] decision. (Burgun 2014)

Input and output randomness are commonly-used terms in contemporary game design, and appear on many taught courses, blog posts and even casual discussion among players. Different games balance input and output randomness in their own unique ways. *Dungeons and Dragons* is an example of output randomness: if a player decides to attack a creature, the attack's success is dependent on a dice roll, the outcome of which cannot be known before committing to the attack. Playing the game therefore requires the player to plan around the possibility of failure and improve their chances of success as much as possible prior to taking an action. Many roleplaying systems incorporate failure as an opportunity for storytelling, thus 'failing' an attempt to do something is as meaningful or important as succeeding.

Magic The Gathering is an example of input randomness: each turn, the player is dealt a selection of cards from a deck of cards they have constructed. In some situations the player may know exactly which cards they will receive (for example, if the number of cards remaining to be drawn is equal to the number that they draw each turn) but in general the player cannot know in advance which ones they will receive. The cards they draw, as well as other forms of input randomness such as the effects of actions their opponents took on the previous turn, will affect what choices they make. Players often employ strategies to limit the unpredictability of their decks, by using cards which create, search for, or copy specific cards and effects.

Input and output randomness are intuitive concepts but quite loosely defined, and there is some overlap between the two concepts. For example, the uncertainty in the output of a player's actions in one moment feeds into the input randomness for the next. If a player fails to successfully kill a monster on this turn, that monster's action will influence their decisions in a future turn. However, not all input randomness effects are sourced from earlier output randomness. For example, some monster attacks in *Slay the Spire* are selected regardless of player activity or randomness, and so act as input randomness but are not random themselves. Random effects are always sourced from some theoretically deterministic source (older games used frame counters and system clocks, while seeded random number generators are more common today) but the player is usually not aware of this source when playing.

In his original blog post, Burgun notes that 'people often use the term "procedural generation" to refer to [input] randomness in digital games'. Burgun is not necessarily equating the two concepts, but does seem to be implying that procedural generation is a type of, or subset of, input randomness. Given Burgun's definition of input randomness, we can see he is arguing that procedural generation informs and influences the player when making in-game decisions.

Defining Procedural Gameplay Systems

From the previous sections we can see that there is a class of generative systems that forms a core of the discussion and research around 'procedural generation' in games, a class that has a distinct impact on a game's *design*, that design-

ers and players perceive differently, and that is linked to the notion of input and output randomness.

Redefining procedural generation as a whole is implausible: for one thing, as any roguelike fan can attest, definitions shift and mould themselves as creative fields change. It is also unnecessary to redraw the boundaries for this term to exclude types of generative system for the purposes of semantics. With that said, however, there seems to be a distinction between 'procedural generation' as it is used colloquially among game-players and critics, and 'generative systems' as a broader field of study and research. Thus, we propose a new term here – *procedural gameplay systems* – to define a suite of techniques aimed specifically at using generative techniques to shape player experience. These systems are a strict subset of generative systems as a whole: all procedural gameplay systems are generative systems, but not vice versa. With this definition we aim to capture the intuition expressed by our survey participants when suggesting a game should be labelled as 'procedurally generated':

> A *procedural gameplay system* (PGS) is a generative system which delivers game content in a manner that the player cannot predict or feasibly enumerate, which changes between play sessions, and which influences the player's decision-making or behaviour.

Where *content* is defined as any element of a game that is represented in data and used during the game's execution (for example, the game's rules, music or levels would count as content, while its metagame, advertising campaign or QA tools would not). The use of SpeedTree to generate scenery in games would not *generally* fall under this definition because, while the content cannot be predicted by the player, it usually does not affect gameplay or player decision-making[3]. Procedurally generated visual effects *might* be argued to fall under this definition, depending on whether we believe the effects are significant enough to alter player behaviour (for example, obscuring the player's view or creating emotional responses such as fear or curiosity).

The generation of levels in *Spelunky*, weapons in *Diablo* or civilisations in *Dwarf Fortress* are all examples of procedural gameplay systems under this definition. All of these examples change the decisions players make either on a micro- or macro-scale. However, the galaxy generation in Elite is *not* a procedural gameplay system – they are the same galaxies for every player, every time they play, and guides to Elite describe each planet and have prepared strategies that can be used in advance. A *Magic: The Gathering* deck is an interesting case in that it acts as a procedural system, but is generally not thought of as such because the player typically designs it themselves. Notably many online tools for designing Magic decks use visualisations similar to those used by procedural generation researchers, showing cost distributions and sample hands. Deckbuilders are an interesting example where gameplay, game design and procedural design all blur together.

'Feasible enumeration' here is broadly construed, but a good guideline is that the player cannot hold all possible outcomes in their working memory. A dice is not a procedural gameplay system in and of itself, although dice tables requiring multiple rolls might be. Drawing a single card from a deck is unlikely to function as a procedural gameplay system, however drawing a hand of five cards from a deck probably does – the possibilities of a single card draw are easily held in mind (52 cards, enumerated in suits and face values), however the possibilities of a five-card hand and the relationships between the cards is much bigger (approximately 2,500,000 unique hands from a 52-card deck). Slay the Spire presents some interesting examples for this definition – most outcomes are not enumerable because the player does not have access to probabilities and distributions, however such data is available on game guide pages and top players often use such information in their runs (Flavell 2022). For most players, most systems in Slay the Spire are PGS.

In the next section we examine how unpredictability in games can manifest through the introduction of PGS, as well as other forms of nondeterminism, and player activity, and how designers leverage and reason about all three of these phenomena to influence player behaviour. We argue that the ways designers engage with these processes are very similar, even if we do not often talk or think about them as such.

Designing for Nondeterminism

In this section we bring together the background discussion of this paper and show how procedural generation and game design both ask a designer to solve the same problem: to create a system that can unfold in many ways, and create a distribution of outcomes that achieves a creative goal.

Suppose we are designing an adventure game, and we have created a room with a treasure chest in it. The chest contains 100 gold pieces, and is protected by a trap that instantly kills the player if they do not disarm it first. This is a precisely-designed low-headroom (in the sense of (Smith 2014)) choice that has a small number of ways the player can interact with it: they can disarm the trap and acquire the contents of the chest, they can fail to disarm the trap and reach a game over state, or they can ignore the chest and continue playing. We have decided to change the design of this treasure chest system to alter how players make the decision of whether to open the chest or not.

Variable Risk

We might decide that a trap that always kills the player is too punishing, or leads to degenerate strategies such as checking every chest. So instead, we set the chest to have a 15% chance to be trapped, and do 10-20 points of damage to the player, who starts the game with 100 health points, if not disarmed. Disarming costs a small amount of some resource (for simplicity, perhaps 10 gold), so players must now decide whether it is worthwhile to check a chest for traps.

There is now a wider range of outcomes than before, and the distribution of these outcomes is affected not only by the features of the trap, but the state of the player when they reach the chest. This demonstrates two important features of design problems such as this: a player experiences a single route through a game, but a designer must reason about

[3]Although its unpredictability is what gives its output an organic appearance, which is why artists and designers use it.

many players' routes; and that any given moment in a game is dependent on both predictable and unpredictable factors. These factors include things the designer knows in advance (the chance a trap will trigger) and things they cannot know in advance (the player's current health when reaching the chest). Many of these unpredictable factors still have limits to their outcomes, for example a player can have between zero and one hundred health points. Other factors are so unpredictable the designer may not be able to model them at all: for example, if the player is tired, distracted or has forgotten that chests in this game can be trapped.

Player-Derived Risk

Now suppose the chest is changed such that when a player opens it successfully, they choose whether it is trapped or not for the next player who tries to open it (perhaps this is an online game so the chest is persistent). A chest now has either a 100% chance to be trapped or a 0% chance to be trapped, but now our ability to model the distribution of events is complicated further, as the probability of a chest being trapped is dependent on player behaviour. Players might always decide to trap the chest, for example, which would result in a situation closer to the original design which encouraged players to check every single chest. A similar metagame to this emerged in the multiplayer component of *Metal Gear Solid V* (Nightingale 2022).

Although we have substantially changed the *distribution* of outcomes, we have not changed any of the outcomes themselves, nor the actions available to the player (save for adding the choice about trapping the chest afterwards). The damage caused by a trap, as well as the reward in the chest, is the same. However the likelihood of a particular outcome occurring is now variable, and player behaviour will influence its future distribution too, as a metagame around chest traps emerges. If we want to shape the behaviour of players when encountering a chest, we will now also need to shape the behaviour of players who have previously opened chests, and what players think about the behaviour of others.

This example highlights two other features of design problems. First, players influence their own game state constantly, as well as those of other players. The next game state the player finds themselves in is a function of their current state and the decisions they make within it (as well as any output randomness). Secondly, designers must contend not simply with the raw mathematics of a given situation, but how players *respond* to it, and player response is not always predictable, rational or explicable. Players might become superstitious about the presence of traps, paranoid about the behaviour of other players, or play at certain times of day to avoid certain groups of players. Designers cannot precisely predict how any given player will behave, but can (and do) attempt to model how groups might behave in general.

Variable Reward

We might observe that the static reward for opening the chest (100 gold) means players develop a fixed assessment of whether it is worth opening a chest or not, regardless of risk. To overcome this, we might put a variable reward in the chest - instead of a fixed amount of gold, we might write an algorithm to procedurally generate an item with one or more random properties, a common system in action-roleplaying games such as *Diablo*. Our generator picks one weapon type, adds two positive effects to the weapon, and one negative effect. For the simplicity of calculations, we will assume the lists of weapon types and effects have ten items each.

This drastically shifts the number of distinct outcomes possible from opening the chest, from our original two (trap activates, or not) to 9,000 possibilities ($2 \times (\binom{10}{1} \times \binom{10}{2} \times \binom{10}{1})$). Earlier in *Variable Risk* we made the distinction between the player, who must only consider the outcomes available in their playthrough, and the designer, who must consider many more paths through the game that other players might take. In this example it is infeasible for a player to calculate each possible item's worth and estimate the expected return. By introducing this procedural reward system, the designer is asking the player to reason about the decision in the same way that they do: to think about spaces and distributions of outputs, rather than single instances.

Over time, a player may gain a sense for the kinds of items yielded by this generator. They might also rely on guides or the advice of other more skilled players to shape their own decision-making. At the same time, just as with the example in *Player-Derived Risk*, the designer cannot know what the player's assessment of the generator will be, or how it will change over time. Different designers will have different ways of modelling player reactions to risk and challenge – including not doing so at all.

Designing for Nondeterminism

In all of the examples we have given above, the game's designer is considering how the structure of a decision point or rule will affect the options available to a player and how the game state evolves in the future. We use examples with numbers and systems here to make some of our arguments more explicit and unambiguous, but the same is true of any type of game design: a dialogue choice in a narrative game; a cosmetic option in a character creator; a menu of gestures and taunts that can be used after beating an opponent.

When we discuss such a choice in a game, we sometimes talk in terms of the actions a player can take, their *action set*. In our first example there are three actions: the player can attempt to disarm the chest; open the chest (possibly triggering a trap); or leave.

However, an action set alone does not describe the possibility space represented by this choice. Facing the chest with 10 health points remaining guarantees the player's death if the trap is set off, while a reward of 100 gold is more meaningful to a player with zero gold than a player with ten thousand. The action set must be considered alongside the current game state, and the decision-making process and risk tolerance of the player. Thus, although a player may view their individual action set as enumerable, the designer considering the same situation views a much larger possibility space, one containing the actions and states not just of one player, but of all players. Changing one aspect of the design usually changes it for all players, and not equally.

From the designer's perspective, we can think of a game

state as a combination of *fixed* and *variable* factors. Fixed factors are things that cannot be affected by the player: the presence of the chest, the probability of it being trapped. Variable factors are those changed by the player: how much health they have left, what other items they are holding, how much gold they have. These fixed and variable factors are then used by the player to make a decision. The designer cannot know in advance how the player will use the information available to them. They might design on the assumption the player is rational, or irrational. They might design to support weaker players, or test more experienced ones. Design decisions take the form of altering the fixed factors (the design of the situation) or the variable factors (the design of the earlier parts of the game that led to this current game state) to yield a particular space of outcomes when combined with an unpredictable factor (player behaviour).

We can see how this maps onto a procedural generator, like the one in the *Variable Reward* example above that generates weapons to put in the chest. A procedural generator can also be thought of as a combination of fixed and variable factors: the structure of the process itself, the code, is fixed; while its parameters are variable and some may be inputs affected by player activity. Procedural generators are driven by a source of randomness – something that the designer cannot know the behaviour of in advance, the unpredictable factor similar to player behaviour.

For both procedural generators and player choice, the space of outputs is defined as a combination of fixed elements (the code of the generator; the choice the player is faced with) and variable elements (the inputs to the generator; the state of the game prior to the player encountering the choice). Applied to this space is some stochastic weighting that the designer cannot control (the random input for a generator; the player's thought process). This combination results in a *weighted distribution* over the outputs: different outputs are more or less likely based on the output of the process. In the player scenario, the player wants to take some actions more or less than others. In the generated scenario, some outputs are more or less likely to be generated.

Designers can change their system – either redesigning part of the game, or rewriting the code of their generator – to alter the space of outputs and, as a result of doing so, the distribution of the weights or probabilities across those outputs. Yet these systems are highly complex, and predicting the effect of a change is extremely difficult. We examine how these changes are often evaluated in the next subsection. The purpose of this section is not to prove the equivalence between game design and generative design, which is impossible without a lot of semantic consensus and formalism, but instead to show evidence that there is significant overlap between the two disciplines, and that this overlap suggests that similar thought processes, practices and solutions might be shared between both areas.

Evaluating Systems

We can see further evidence for the parallels between generative systems design and game design by considering how both of these systems are evaluated by designers. In the previous subsection we observed that the complex relationship between fixed, variable and random elements in both types of system means that designers can only exert indirect influence on the outputs/outcomes, by changing the structure that the randomness is filtered through and how it affects the distribution of the potential space of outcomes. This makes evaluation a critical part of designing both kinds of system.

By far the most popular way of evaluating procedural generators is through sampling their output. This technique is used in the game development community as well as being the basis for the most popular academic technique, expressive range analysis (Smith and Whitehead 2010), although most developers using generative techniques do not sample at scale and instead manually test and evaluate 'by eye'. In our survey of 126 game developers, 97% of developers tested procedural generators through playing their game, and 83% tested through repeatedly generating outputs in an editor, while only 26% automated this process, and only 7% used expressive range analysis. Sampling by eye only shows a small snapshot of the distribution of generated outputs, but it can support the designer in understanding their generator.

Equivalently, one of the most important ways to evaluate a game is through playtesting. One of the goals of playtesting is to understand how a range of players react to the current state of the game's design – in other words, to sample from the set of all possible players (by choosing playtesters), and then sample individual playstyles (by having them play the game). These players represent only a small sample of the target audience for a game, but they help provide an updated estimation of how players might respond in the scenarios the game is currently presenting to them.

A key difference is in the nature of the sampling process for both exercises. For generative systems that are driven through conventional random sources such as noise functions, we can easily test our generators with the exact stochastic sources it would use during normal operation. This means that given enough time we can easily approximate the behaviour of the system as a whole (or even enumerate the entire space, as in (Sturtevant and Ota 2018)). Playtesting is less straightforward, because humans are not well-behaved (in any sense of the phrase), their behaviour is changeable and the selection of playtesters interacts in complex ways with other aspects of game production such as marketing. If we take a biased sample of playtesters, or we fail to get a sample that is representative of our target audience, then our evaluation will give us a false idea of the distribution of player preferences who will likely play our game on release.

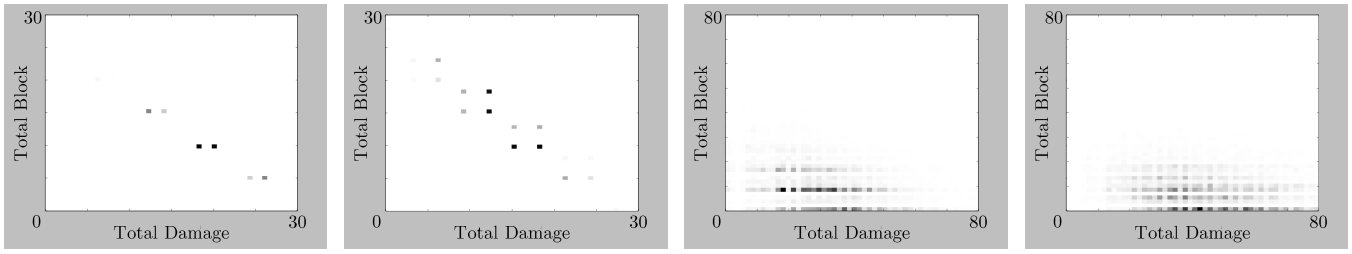
Discussion

If game design and generative design are two sides of the same coin, what new research questions does this open up to us? What new knowledge can we gather from this insight?

Knowledge Exchange

If we accept that game design is a form of generative design, then this opens up many opportunities to approach the analysis of other game systems through a generative lens.

For example, deckbuilding games such as *Slay The Spire* task a player with creating a deck of cards through the course



(a) Ironclad Starter Deck (No Modifications)

(b) Silent Starter Deck (No Modifications)

(c) Silent deck, Ascension 0, won (Floor 51)

(d) Silent deck, Ascension 0, lost (died, Floor 50)

Figure 1: Expressive range analyses of opening hands of various Slay the Spire decks. The x-axis shows the sum of total damage in the opening hand, while the y-axis shows the sum of block values (incoming damage mitigation).

of a game which they shuffle and draw from to solve challenges. If we consider the player's deck to be a procedural generator, we can use techniques from the analysis of procedural generators to assess the qualities of its generative space and assess the performance of players or compare the design of characters. Figure 1 shows an *expressive range analysis* (Smith and Whitehead 2010) applied to the starting decks of two Slay the Spire characters. In each case, we randomly shuffle the deck, deal a hand of five cards, and calculate the total attack damage and total damage block amount afforded by the five cards. We plot this on a histogram and repeat the process 100,000 times. Using expressive range analysis we can easily visualise the behaviour of the different decks – the Ironclad, focused on dealing damage, has less variation in blocking value (because it only contains one type of block card), and the Silent, a more complex character that can deal damage in indirect ways, offers less damage on average (because it mostly only has a single type of damage card).

We can perform a similar analysis of a more complex deck, built by players at the end of the game. The second two images in Figure 1 shows an expressive range analysis of two Slay the Spire decks for the Silent character, again showing total damage and block content from 100,000 possible opening hands. We analyse one deck which won, and one deck which lost. Both decks have been modified heavily by players from the starter decks over the course of the game. We can see from the two figures that the winning deck has less damage in the opening hand, and more consistently reliable block, whereas the losing deck is more often dealt a hand with no defence cards in at all. This surface-level analysis does not tell us everything we need to know about the play sessions, but they are an interesting additional tool for studying player strategy spaces. Such analyses could be used to interpret player data, analyse automated playtesting, trial new features or prepare for patches and bug fixes. The generative nature of gameplay is a natural fit for analytical techniques designed for generative systems.

In (Sturtevant 2021) we can see an example of procedural generation research directly being applied to non-procedural game design. Sturtevant generates *all* possible levels for a puzzle game and then shows this can be used as a database to perform search for levels with specific properties, as the basis of an ingame level generator. However, Sturtevant also shows this database can be used to give insight into properties of the design space (analysing the difficulty of a level versus its size) and even form the basis of design tools, where a human designer can receive real-time hints about the number of possible solutions to a level, for example, as they place level elements. Here, the equivalence of generative thinking and game design is made explicit, and creates a new way of designing levels.

Play as Space Exploration

Throughout this paper we have discussed the concept of 'unpredictable' game content, and tied this notion strongly to the definition of procedural generation. Yet for a player approaching a game or challenge for the first time, many things about their experience are unpredictable whether or not the designer intends them to be.

If we consider two players, one experiencing Level 1-1 of *Super Mario Bros.* for the first time, and one experiencing the first floor of *Spelunky* for the first time, they both have a broadly similar experience in terms of their relationship with the game they are playing. They are unfamiliar with the level layout, enemy behaviours, item functionality and character control schemes. Neither game appears 'more' procedural or random than the other. The *second* time each player plays the same game section their experiences differ considerably. The Mario player sees the same level geometry, and the only change they experience is caused by their own inputs to the game. The Spelunky player sees entirely new level geometry, thanks to the procedural level generator, and may see systemic interactions between enemies, items and the environment that they did not cause themselves and have not seen before.

In *Designing for Nondeterminism* we described spaces of states and actions from the perspective of the designer, but not from the perspective of the player. To the player, their understanding of the state and action space is always incomplete. They can never know if they have a full picture of the actions available to them, nor can they know if they understand all states the game may be in or all transitions between them (even for 'simple' games (Droqen 2013)). They may suspect or assume the presence of nonexistent states – consider a player who has recently played a different game with *mimics* in, monsters disguised as chests. They may be-

lieve that the game in our earlier examples could also contain mimics, and avoid opening any chest as a result.

Thus we might consider single-player, perfect-information games such as Sudoku as games that are unpredictable in their own way, with nonuniform spaces of states and transitions. As a player fills in a Sudoku they do not know what new possibilities each number will open up, and the rate of new branches opening up will change as they work, along with the areas of the grid they work in. The player does not think of this experience as unpredictable, but each player solves a Sudoku in a slightly different way, at a slightly different speed, with mistakes, notes and guesses unique to them. Designers account for this, consciously or otherwise, as they tweak their designs.

This is not to say that Sudokus are roguelikes. It is to say, however, that the unpredictable nature of play (and players) extends to every type of play and game, even the most logical, rational, step-by-step problem solver. Game designers embrace and work with this stochastic texture in an amazing variety of ways, and understanding this work as similar to generative systems design will not just affect systems-heavy games (as we used in our simple examples) but games of all kinds - narrative, improvisational, party, rhythm, logic and beyond. Games are about designing predictable systems, and then letting unpredictable players loose on them.

Bridging the Divide

Before the onset of 'generative AI' as a buzzword in the technology and creative industries, procedural generation was already a somewhat polarising topic. Many talks about the topic dwell on its drawbacks, even those ostensibly promoting it (Duncan 2015). Drawing connections between game design and the design of generative systems helps us examine some of these criticisms in a new light, and open up new avenues we can explore to overcome these limitations and support new creators – to embrace procedural generation as a practice that we can grow and celebrate, rather than a technical discipline for programmers to churn out 'content'. This is not to say that our goal should be to make everyone use procedural generation; simply that critics often reveal to us things we can fix or change for the better.

Procedural generators are often described as 'too random' by designers, which is an indirect way of expressing that they feel a loss of authorial control when replacing hand-designed content with generated content. We know that games include many nondeterministic and unpredictable systems even without procedural generation in them – branching narrative paths, dice rolls and player activity all make it hard for a designer to know what routes through a game system exist. Why, then, do designers feel differently about some kinds of nondeterminism or unpredictability, and what might this tell us about alternative routes we could explore in generative research?

The page limit for this paper does not permit a broader discussion on this topic, but there are several factors we believe affect this. One is that that player behaviour is less variable than a random number generator and designers often believe (rightly or wrongly) that they are able to approximate or estimate the behaviour of their target audience – their ability to predict players is not only rooted in their experience as designers, but their own experience as a player. If we consider players more predictable than generators, it may be worth studying generative systems with intentionally restricted spaces of outputs – orders of magnitude less than normal – to see what textures and dynamics this offers us. Minecraft, for example, can generate up to 2^{64} possible worlds. If the generator could 'only' produce 2^{10} worlds at most – either through an artificial filtering of its generative space post-hoc, or a pre-emptive restructing of its algorithm – what alternative routes for analysis, design and control does this offer? Studies into both player perception of variation, and designer perceptions of control, might reveal whether scale has been a self-defeating goal for generative design in games.

Conclusions

In 2025, the award-winning game design Jon Perry wrote:

> *Game design is so much about coralling uncertainty. Constructing a little pen where uncertainty can prowl around and delight onlookers but never actually escape the confines.* (Perry 2025)

Game design *is* generative design. This is not simply a catchy paper title: we have argued and shown that the distinction between 'procedural content' and 'hand-designed content' is academic at best, and that most games include some aspect of unpredictability that designers must either anticipate, control or work around. Unpredictability is the essence of many, many types of game – it is surprising, challenging, joyful, frustrating, confusing and playful.

In this paper we have discussed the current understanding of procedural generation in games and in games research, and connected this to a broader idea of the role of randomness and unpredictability in games. To fit this, we defined a new subspace of generative system: *procedural gameplay systems* which produce game content which the player cannot predict or enumerate, but which influences their in-game decision-making. We showed several examples of game systems where this definition can be applied, some in obvious ways, others less so.

For some people, generative AI heralds an era where people are no longer necessary to design games. For others, even the thought of using random generators for level design relinquishes too much of their authorial control. Our hope is that we can show that what we call *generative literacy* – the creative ability to understand spaces of experience and possibility, and to make design decisions to shape them – is a fundamental skill in game development, and that by nurturing a better understanding of this artistic skill we can help teach it to others, build connections across domains, and push the limits of this medium, further than ever.

Acknowledgements

Thank you to the reviewers who gave feedback on this unusual paper, both at AIIDE and other venues. Thanks also to everyone who ever entered PROCJAM or are part of its community, who have inspired and taught me so much.

References

Block, A.; Oakley, P.; and Swanepoel, M. 2022. Procedural music generation with Quartz. https://www.unrealengine.com/en-US/tech-blog/procedural-music-generation-with-quartz.

Brewer, D. 2021. Managing Pacing in Procedural Levels in Warframe. In Rabin, S., ed., *Game AI Pro 4*, chapter 7, 1–13. CRC Press.

Burgun, K. 2014. Randomness and Game Design. Archived on the Wayback Machine: tinyurl.com/burgunrandom.

Burke, C. 2019. Does Borderlands 3 have too many guns? Archived on the Wayback Machine: tinyurl.com/borderlands1b.

Bycer, J. 2015. Procedural vs. Randomly Generated Content in Game Design. Archived on the Wayback Machine: tinyurl.com/gd-proc-vs-random.

Connor, A. M.; Greig, T. J.; and Kruse, J. 2017. Evaluating the Impact of Procedurally Generated Content on Game Immersion. *The Computer Games Journal*, 6(4): 209–225.

Doull, A. 2008. The Death of the Level Designer. PCG Wiki.

Droqen. 2013. Block Faker. droqen.itch.io/block-faker.

Duncan, G. 2015. The Art Direction of No Man's Sky. tinyurl.com/gdcnms.

Flavell, S. 2022. RECORD STREAK? 15-0? I think it's actually possible to lose this. Available on the Internet Archive: https://web.archive.org/web/20220316092227/https://www.youtube.com/watch?v=6-8DzH_JEIw. Timestamp: 30:00.

Frontier Developments. 1984. David Braben and Ian Bell.

Games, H. 2016. No Man's Sky.

Gouskos, C. 2006. Will Wright Gives the Best Speech I've Ever Heard. Archived on the Wayback Machine: http://tinyurl.com/gdc-2005-wright.

Karth, I.; and Compton, K. 2023. Conceptual Art Made Real: Why Procedural Content Generation is Impossible. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*.

McKendrick, I. 2014. Procedural Doesn't Mean Random - PROCJAM Talk.

McMillen, E. 2011. The Binding Of Isaac.

Metanet Software. 2015. N++. Archived on the Wayback Machine: tinyurl.com/nplusplusgame.

Moby Games. 2020. Baldur's Gate 3 - Credits. https://www.mobygames.com/game/150689/baldurs-gate-iii/.

Mojang. 2011. Minecraft.

Mossmouth. 2010. Spelunky.

Nightingale, E. 2022. Turns out Metal Gear Solid 5's nuclear disarmament was rigged all along. Eurogamer.

Perry, J. 2025. Game design is so much about coralling uncertainty... bluesky. Archived on the Wayback Machine: https://tinyurl.com/perry-unpredictability.

Shaker, M.; Sarhan, M. H.; Al Naameh, O.; Shaker, N.; and Togelius, J. 2013. Automatic generation and analysis of physics-based puzzle games. In *Proceedings of the Conference on Computatonal Intelligence and Games, CIG*, 1–8.

Smith, A. 2014. Strategy headroom in roguelikes. http://tinyurl.com/smithheadroom.

Smith, G. 2015. An Analog History of Procedural Content Generation. In *International Conference on Foundations of Digital Games*.

Smith, G.; and Whitehead, J. 2010. Analysing the Expressive Range of a Generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*.

Sturtevant, N. 2021. An Argument for Large-Scale Breadth-First Search for Game Design and Content Generation via a Case Study of Fling! In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Sturtevant, N. R.; and Ota, M. J. 2018. Exhaustive and semi-exhaustive procedural content generation. In *Proceedings of the Fourteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Togelius, J.; Champandard, A. J.; Lanzi, P. L.; Mateas, M.; Paiva, A.; Preuss, M.; and Stanley, K. O. 2013. Procedural Content Generation: Goals, Challenges and Actionable Steps. In *Artificial and Computational Intelligence in Games*, volume 6 of *Dagstuhl Follow-Ups*, 61–75.

Zafar, A.; Mujtaba, H.; and Beg, M. O. 2020. Search-based procedural content generation for GVG-LG. *Applied Soft Computing*, 86.