arXiv:2305.02131v1 [cs.AI] 3 May 2023

为什么燕麦片很便宜：柯尔莫哥洛夫复杂度与程序化生成

YOUNÈS RABII[☿]，伦敦玛丽女王大学，英国
MICHAEL COOK[Ψ]，伦敦国王学院，英国

尽管程序化生成在游戏开发者中颇受欢迎，但关于这一主题的学术研究主要聚焦于新的应用，也有一些研究涉及实证分析。本文将信息论中的理论工作与游戏内容生成联系起来。我们证明，生成器所能生成的最复杂作品的柯尔莫哥洛夫复杂度，与该生成器的可能性空间大小之间存在关系。由此，我们指出生成器中编码的知识、其输出空间的密度，以及它所生成作品的精巧程度之间存在制约关系。我们将这一结果与资深程序化生成器设计者的经验联系起来，并通过一些示例加以说明。

CCS概念： •**软件及其工程** → **交互式游戏**； •**计算数学** → **信息论**。

ACM参考文献格式：
YounèsRabii 和 Michael Cook。2023年。燕麦粥为何便宜：柯尔莫哥洛夫复杂度与程序化生成。载于 数字游戏基础会议 *2023*（*FDG 2023*），*2023*年*4*月*11*日至*14*日，里斯本，葡萄牙。ACM，纽约，NY，美国，11 页。
https://doi.org/10.1145/3582437.3582484

1 引言

程序化内容生成已成为现代游戏开发者工具箱中一个成熟的组成部分。游戏产业规模最大的活动——游戏开发者大会——在过去十年中举办了 50 多场关于程序化生成的演讲，演讲者既包括小型独立开发者，也包括大型 AAA 公司，涵盖从程序设计到艺术再到写作等多个领域。与此相应，在过去二十年中，程序化生成日益成为游戏人工智能研究者关注的热门话题。FDG 的程序生成研讨会如今已进入第十二年，是游戏人工智能领域历史最悠久的研讨会之一；而各类会议中设置专门的论文专题也已成为惯例。

尽管内容生成极其重要，且人们投入了大量时间开发实用技术，但对程序化生成器的分析仍是一个相对不成熟的研究领域。过去二十年的研究中已经出现了一些值得注意的技术 [7, 8], 以及对其有效性的研究 [4, 9], 但游戏研究人员所使用的许多技术在这段时间里几乎没有变化。因此，大量程序化生成工作都是凭借“感觉”完成的，并通过 Roguelike Celebration 等活动分享事后总结[1] 这些总结表明某种方法是成功的，其他人可以尝试复现或在其基础上进一步发展。

尝试对游戏开发的知识进行抽象和概括十分重要，因为这有助于我们在不同的游戏、设计师和技术之间建立联系。它们还有助于为有关人工智能的“民间传说”或许多游戏开发者都提及、却从未得到

[☿]提及该作者时，必须使用 they/them 单数中性代词。[Ψ]提及该作者时，必须使用 he/him 阳性代词或 they/them 单数中性代词。[1]https://www.roguelike.club/

允许出于个人或课堂使用目的，免费制作本作品全部或部分内容的数字副本或纸质副本，但前提是不得为营利或商业利益制作或分发副本，且副本首页必须载有本声明及完整引文。对于由作者以外的其他人拥有版权的本作品组成部分，必须尊重其版权。注明出处后可以进行摘要。除此之外，如需复制、再版、发布到服务器或重新分发到列表，必须事先获得具体许可和/或支付费用。请通过 permissions@acm.org 申请许可。© 2023年 版权所有归作者/权利所有者所有。出版权已授权给 ACM。稿件已提交至 ACM

具体证实的现象提供依据和证据，甚至加以证明。理解这些感受从何而来，并将其与计算机科学中的既有观点联系起来，能够带来新的发现，也能加深我们对这门技艺的理解。

本文将柯尔莫哥洛夫复杂度这一源自信息论的概念与游戏内容生成联系起来。我们证明，生成器所能生成的最复杂作品的柯尔莫哥洛夫复杂度，与该生成器可能性空间的大小之间存在固定的关系。由此，我们揭示了生成器中编码的知识、其输出空间的密度以及所生成作品的复杂程度之间的限制性关系。我们将这一结果与程序化生成器领域的常见说法联系起来，并通过示例加以说明。

2 背景

2.1 柯尔莫哥洛夫复杂度

信息论研究信息如何被传递和表示，其起源早于编程语言的发展。信息论常用于分析程序和计算，尤其是作为复杂性理论和可计算性分析的一部分。

在 [3] 作者引入了算法复杂度这一概念，后来被称为柯尔莫哥洛夫复杂度。一个对象的柯尔莫哥洛夫复杂度定义为能够在执行时生成该对象的最短程序的长度。例如，一个由一千个零组成的字符串可以写成如下形式：

```
对于 i 在范围 ( 0 , 1 0 0 0 ) :打
印 ( 0 )
```

然而，一个包含前一千个质数的字符串需要更复杂的程序才能计算出来。一般而言，对象越难压缩，其柯尔莫哥洛夫复杂度就越高，因为要描述对象的每个部分，需要更具体的代码。

2.2 生成系统理论

生成系统的分析是游戏人工智能领域以及更广泛的数学空间中的一个研究课题。与建模、预测和概率相关的领域，都与预测复杂系统或非确定性系统的行为存在一定联系。

更具体地说，在游戏人工智能领域，研究人员已经开发出用于理解程序化生成器行为的技术，通常旨在构建能够为设计师和开发者分析并可视化这些信息的工具。一个突出的早期例子是 Smith 和 Whitehead 提出的表达范围分析（ERA），其中他们反复对内容生成器进行采样，然后将采样内容的度量数据绘制在直方图上[7]。只要谨慎选择度量指标，这就提供了一种直观理解生成器行为的方法。

Summerville 在 [8] 中基于这项工作进一步展开，并提出了扩展该方法的途径，以采用更丰富的可视化方式，尤其是针对基于机器学习的程序化内容生成（PCGML）领域，因为该领域需要采用不同的评估方法。Cook 等人在 [2], 中也进一步拓展了 ERA，构建了一种辅助设计工具，通过执行随机化分析，展示生成器自身设计空间的元层级探索。他们的工具 Danesh 还提供了多种智能方法，用于调整和修改生成器，同时考虑到生成系统适应度景观的不均匀性以及参数的非线性行为。

用于评估内容的指标选择也是研究主题。分析技术通常具有通用性，但仍依赖特定指标来提供特定领域的背景信息，从而更深入地理解特定生成器输出的质量。这是内容生成分析的一大弱点，因为撰写有用的

指标是一项难度很高的技能，需要深入了解应用领域。对这些指标的分析表明，它们在预测质量方面的效果参差不齐；即使是针对特定体裁的指标，其通用性也尚不明确 [4]。

本节列出的所有方法本质上都是经验性的，需要对生成器的输出进行实验分析。这些方法为开发者应用于自己的系统提供了实用且有效的技术。本文试图通过提出一个立足于生成系统基础理论的结论，补充这方面的研究成果。拓展这项研究有助于更深入地理解所有生成程序的运作方式。

2.3 民间智慧与程序化生成

或许部分归因于程序化生成作为一种与实验和跨学科研究相关的技术，吸引了来自学术研究、艺术、游戏产业等领域的众多实践者，形成了一个活跃的社群。过去十年间，许多非官方、非正式的活动和社群相继涌现，致力于生成式软件，尤其是游戏领域，例如 PROCJAM[2] 和 Everything Procedural[3]，以及像 Roguelike Celebration 这样重点关注程序化内容生成（PCG）的活动。围绕这项独特技术分享实践和经验，逐渐形成了一种关于程序化生成的民间智慧，其中融合了幽默、实践经验与内化的知识。

一个著名的例子是燕麦粥的万碗难题，这个术语由 Kate Compton 创造，如今已成为程序化生成从业者中最广为人知的说法之一。在这个类比中，Compton 将程序化生成的内容比作一碗碗燕麦粥，并借此指出，程序化生成相关的推销话术常常强调多样性或不可预测性，但这些说法其实毫无意义。Compton 解释说，每碗燕麦粥都独一无二，但这并不意味着它们有趣或有价值。设计师借此认识到，仅靠程序化生成并不能保证内容丰富多样或引人入胜；必须精心设计系统，将生成方法用作表达工具，而不是把它本身当作解决方案。我们不难想象，也可以用一种不那么引人入胜、令人难忘的方式表达类似的观点，尤其是在学术研究中——学术研究常因过于刻板而受到批评。这个比喻的巧妙之处，无疑是让这一观点广为流传并被人铭记的关键。

程序化生成的从业者也会深入探讨使用这项技术时遇到的挫折与困难。游戏开发者 Orteil 曾发布一条广为传播的推文，写道：“多亏了程序化生成，我可以用双倍的时间制作出双倍的内容”[4]。这句带着自嘲意味的话让我们对程序化生成社群有了不少了解：他们有自知之明，明白人们围绕这项技术编织了哪些神话；即便如此，推文作者仍然乐于使用 PCG。

本文的目标之一，是将关于生成系统与编程的形式化理论思想，与程序化生成从业者的直觉和内化知识联系起来。通过这样做，我们希望能够深化这些思想，帮助程序化生成领域的社区在此基础上继续发展，同时鼓励该领域开展更多学术研究。

[2] https://www.procjam.com/ [3] https://everythingprocedural.com/ [4] https://tinyurl.com/orteilpcg22

3 证明：生成系统的复杂性与局限

本节将给出一个证明，阐明生成产物的柯尔莫哥洛夫复杂度、生成它们的算法，以及该算法所定义的可能性空间之间的关系。我们首先定义证明中使用的术语，以这些术语陈述定理，然后描述证明本身。

定义 *3.1*。 程序是一个有限二进制字符串， $p \in \{0,1\}^n$ 适用于 $n \in \mathbb{N}$。输入表是一个有限二进制字符串，$i \in \{0,1\}^n$ 适用于 $n \in \mathbb{N}$。制品是一个有限二进制字符串， $a \in \{0,1\}^n$ 适用于 $n \in \mathbb{N}$。

定义 *3.2*。 $|p|$ 表示有限二进制字符串 p的长度。$\#S$ 表示有限集合 S所包含的元素数量。

定义 *3.3*。 生成器 G 是一个确定性程序，可建模为有限函数，将输入表 i 映射为制品 a。 G的可能性空间（即由 G 输出的所有制品构成的集合）记为 $\pi(G)$。

$$\pi(G) = \{G(i) : i \in \{0,1\}^n, n \in \mathbb{N}\}$$

定义 *3.4*。如果一个程序会终止，并在有限时间内返回输出，则称其为终止程序。如果一个生成器 G 会终止，且满足以下属性，则称其为理想生成器：

- 固定输入长度： G仅接受固定长度的二进制字符串作为输入，记为 $input_G$。
- 单射性： G是一个单射函数（也就是说，每个长度为 $input_G$ 的输入都恰好对应一个输出，并且不同的输入对应不同的输出）。

请注意，大多数程序化生成器都会终止，尤其是那些用于生成游戏内容的生成器（无论在线还是离线）。然而，理想生成器的另外两个属性并非如此：固定输入长度和单射性。尽管如此，任何非理想生成器都可以直接转换为理想生成器。我们将在附录A节中介绍这一转换。

从定义3.4中理想生成器的后两个属性可以看出，生成器的可能性空间大小与其输入的固定长度直接相关：

$$\#\pi(G) = 2^{input_G} \tag{1}$$

这是因为每个输入都必须对应唯一的输出（由单射性保证），并且 G 必须接受所有长度为 $input_G$ 的二进制字符串，不接受其他字符串。

定义 *3.5.* 柯尔莫哥洛夫复杂度， K是一个函数，它以制品 a为输入，并输出最短的组合程序和输入的长度，分别为 p 和 i，使得 $p(i) = a$。

$$K(a) = \min_{p,i:p(i)=a} |p| + |i| \tag{2}$$

K^* 是一个函数，它以生成器 G为输入，并返回 $\pi(G)$中任意制品的最大柯尔莫哥洛夫复杂度。也就是说：

$$K^*(G) = Max\{K(a) : a \in \pi(G)\} \tag{3}$$

在下面的定理中，我们给出了理想生成器 G条件下 $K^*(G)$ 的下界和上界。随后，我们将说明这一结果为何与现代程序化生成理论相关，并举例说明其应用。

定理 3.6。理想生成器 G 始终满足以下不等式：

$$|G| + log_2(\#\pi(G)) \geq K^*(G) \geq log_2(\#\pi(G))$$

证明。任取一个生成器 G，使得 G 是理想生成器。我们分别证明每个不等式。

上界: $|G| + log_2(\#\pi(G)) \geq K^*(G)$。
根据柯尔莫哥洛夫复杂度的定义：

$$\exists i \in \{0,1\}^{input_G} . |G| + |i| \geq K^*(G) \tag{4}$$

S由于 G 是理想生成器，因此对于 G定义域中的所有输入， $|i|$ 都相同，即 $input_G$。因此 ：

$$|G| + input_G \geq K^*(G) \tag{5}$$

由 1 可得：

$$input_G = log_2(\#\pi(G)) \tag{6}$$

由 5 和 6，我们可以推导出 $|G| + log_2(\#\pi(G)) \geq K^*(G)$ 如所需。
下界： $K^*(G) \geq log_2(\#\pi(G))$。
根据柯尔莫哥洛夫复杂度的定义，对于 $\pi(G)$ 中的每个制品 a，至少存在一个确定性程序 p_a，其输出为 a，且满足 $|p_a| = K(a)$。令 P_a 为所有此类程序构成的非空集合。根据 P_a 的定义，我们知道 P_a 中的程序数量至少与 $\pi(G)$ 中的制品数量相同，即 $|p_a| \geq \#\pi(G)$。因此，我们有：

$$log_2(|p_a|) \geq log_2(\#\pi(G)) \tag{7}$$

P_a 是一个有限二进制字符串构成的非空集合。根据鸽巢原理，至少存在一个 $p \in P_a$ 使得：

$$|p| \geq log_2(|p_a|) \tag{8}$$

由 7 和 8，以及 $\geq$ 运算符的交换律，我们推导出：

$$|p| \geq log_2(\#\pi(G)) \tag{9}$$

根据 K^* 在3中的定义，可得：

$$K^*(G) = Max\{|p| : a \in \pi(G), p \in P_a\} \tag{10}$$

由9 和10，可得：

$$K^*(G) \geq log_2(\#\pi(G)) \tag{11}$$

□

4 讨论与启示

在上一节中，我们概述了一项证明，该证明运用柯尔莫哥洛夫复杂度，将生成系统的不同属性联系起来，即其可能性空间的大小（$log_2(\#\pi(G))$）、其最复杂制品的柯尔莫哥洛夫复杂度（$K^*(G)$），以及生成器代码的长度（$|G|$）。本节将这一结果与程序化生成实践的相关方面联系起来，并讨论其对 PCG 研究的启示，从而为这一结果提供背景。

4.1 生成式设计中的权衡

通过将上述证明中的各个要素与设计程序化生成器时较为通俗、日常的方面联系起来，我们可以开始把证明结果与对生成系统局限性的直观理解联系起来。

4.1.1 编码知识。程序化生成器的设计通常涉及研究、实践和实验。例如，要设计一个能够程序化生成诗歌的系统，生成系统设计师可能会阅读有关诗歌理论及诗人所用技巧的文章，尝试自己写诗，或者大量阅读自己感兴趣并希望复现其风格或流派的诗歌。这种深入研究并非总是必要的——设计师可能会决定以自己已有的诗歌知识为基础开展工作，也可能本身就是一位经验丰富的诗人，已有多年的创作实践。即使在研究或实践不适用的情况下，例如将程序化生成用作压缩或随机化的一种形式，仍然需要谨慎规划，思考生成系统的分布、功能和目标。

编写程序化生成器时，设计师会将自己对相关生成问题的知识嵌入所编写的代码中。希望嵌入系统的知识越多，需要编写的代码就越多。例如：用于处理希望从可能性空间中排除的边缘情况的代码；用于描述特定形式或结构模板的代码；或者用于描述噪声或随机性的特定分布、以提供恰当纹理基础的代码。我们可以将生成器代码的长度（$|G|$）视为编码进生成器的设计知识的近似指标。请注意，$|G|$ 表示会影响生成器功能的最少代码，而不是对任意代码进行计量。因此，添加空语句不会增加 $|G|$，但添加会影响内容生成方式（从而影响其柯尔莫哥洛夫复杂度）的代码则会计入。

4.1.2 规模。 主打程序化生成的游戏，其营销内容也可能会提及可能性空间的规模或大小，例如《无主之地3》的营销活动就强调，游戏中有“超过十亿种枪械”[5]。早期许多支持在游戏设计中使用程序化生成的论点，都源于人们认为它能够为玩家创造“可重复游玩”或“无穷无尽”的内容。尽管这种技术在某些用途上的确能做到这一点，但规模并非总是必要的，也不总能保证质量或适用性。

在我们的证明中，可能性空间的大小（$log_2(\#\pi(G))$）体现了生成器能够创建的潜在输出数量。潜在输出数量较少可能表明生成器缺乏多样性，这可能意味着内容体验会受到重复的影响。或者，生成器也可能被设计用于创建一组特定的输出（例如在《Elite》中使用 PCG-as-compression [1]），或者生成器被有意保持较小，以便玩家能够学习或预测其行为。同样，潜在输出数量较多也可能意味着一个由非常相似的输出构成的单调空间（第 燕麦粥 问题中提到的2.3节）。或者，它可能被用来传达广阔感和重复性（正如 Emily Short 所建议的 [6]），也可能由足够多的编码知识提供支持，从而即使在大规模下也能保持多样性。我们将在下一节中更详细地讨论燕麦粥问题及其与我们证明的关系。

4.1.3 模式密度。 随着时间推移，玩家会自然而然地学会识别游戏内容中的模式。这并非程序生成内容所独有；例如，玩家经常会抱怨游戏在多个区域重复使用素材，或是辨认出某款游戏引擎特有的外观或感觉。然而，由于程序生成内容具有算法性质，因此更容易被玩家以这种方式识别出模式。如果生成内容中的模式太容易被察觉，玩家可能会认为它“重复”。

[5]https://www.youtube.com/watch?v=bFLhcoFAJMQ

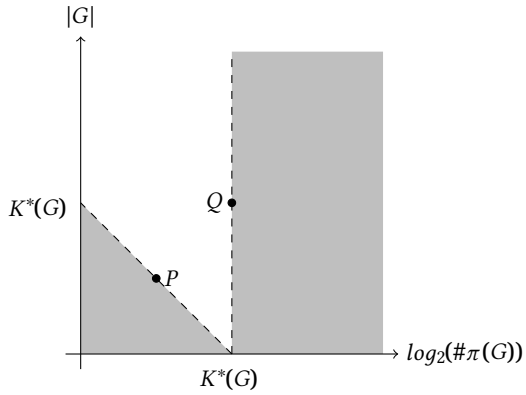

图 1. $|G|$（生成器的长度）与 $log_2(\#\pi(G))$（可能性空间的大小）之间关系的图示。标记为 $K^*(G)$ 的交点表示 $\pi(G)$ 中最复杂作品的柯尔莫哥洛夫复杂度。

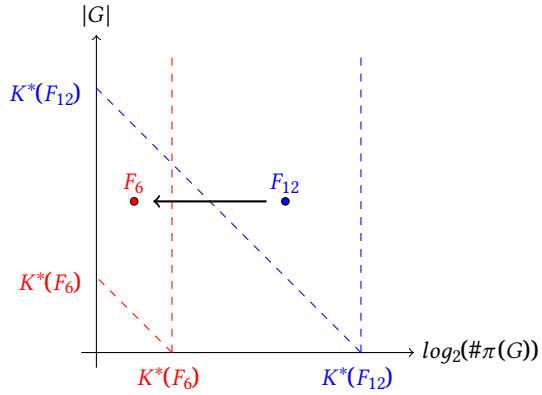

图 2. 图 1 的重新绘制图1，其中的标记指示了花朵生成器的两个版本，F_{12} 和 F_6。

延缓玩家学会识别模式，有许多方法。一种方法是单纯提高模式密度 ——在生成过程中添加更多细节和模式，让玩家更难记住上次遇到某个设计元素是什么时候。另一种方法是加入干扰元素，模糊不同模式之间的界限。《Spelunky》便采用这种方法，将离散的模板与更连续的随机性融合起来。

柯尔莫哥洛夫复杂度表示描述某个特定对象所需的程序代码量。制品越复杂、越嘈杂、越详细或越随机，描述它所需的代码就越多。可能性空间中最 复杂的制品（在我们的证明中记作 $K^*(G)$）代表特定生成器在这一属性上的上限。该值越高，生成器能够产生的复杂度就越高。

4.2 与定理的关系

结合这些定义重新审视证明的结果，我们可以看出，生成算法中编码的信息量、算法能够生成的事物数量，以及其输出中模式和细节的复杂程度之间存在关联。由于这些概念在我们的定理中彼此相连，改变其中任何一个都会影响其他概念的值。

为了直观呈现这一点，图 1 绘出了定理中关于生成器 G的两个不等式。根据我们的定理，$|G|$和 $log_2(\#\pi(G))$的值受这些直线的约束，这意味着 G始终会落在图中的非阴影区域。因此，尝试改变其中一个值，可能也需要同时改变其他值。例如，看看图中的生成器 P。要缩短它的代码长度（$|G|$），我们必须降低 $K^*(G)$，或者提高 $log_2(\#\pi(G))$，因为这受到定理所给出的界限约束。用本节的说法来说，我们无法从生成器中移除知识，除非降低它所能生成内容的模式密度，或扩大其输出的尺度。同样，对于 Q，提高 $log_2(\#\pi(G))$就需要提高 $K^*(G)$。换言之，要扩大生成器输出的尺度，我们也必须提高它所创建内容的模式密度。下一节将通过示例，进一步深入探讨程序化生成器这些属性之间的不变关系。

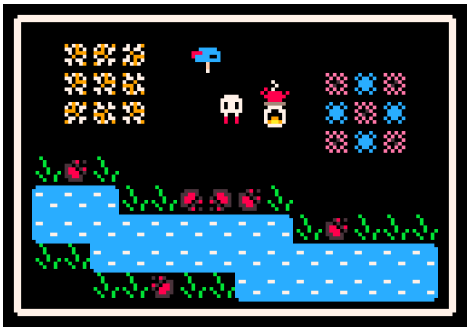

图3. 茶园的开发过程截图。地图右上方和左上方，以及底部河流附近，都可以看到成组生成的花朵精灵图。

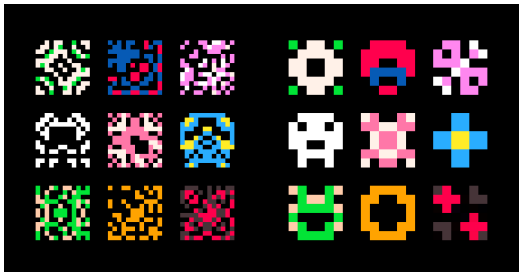

图4. 茶园开发过程中生成的 12x12（左）和 6x6（右）花朵精灵生成器输出样例。设计师选择只在最终游戏中保留 6x6 制品。

5 示例

5.1 花朵生成器

茶园 是一款即将推出的独立开发电子游戏，讲述探索梦境世界的故事。在游戏中，玩家可以采摘花朵来泡茶；饮下茶后，便会梦见开满鲜花的花园。玩家可以采摘一朵花，将它从梦境带回现实世界，再用它泡制更多茶。游戏广泛运用了程序化生成，尤其用于生成梦中花园的布局以及花朵本身的造型。图 3 展示了游戏中的一张截图。

设想一个程序化内容生成器，它能为《茶园》生成花卉图案。生成器的输出始终是一个 12x12 的二维像素颜色数组。我们将这个生成器称为 F_{12}，其中的数字下标表示花朵以像素为单位的宽度和高度。现在来看这个生成器的一个修改版本 F_6，它与原版完全相同，唯一区别是花朵的尺寸变为 6x6。生成器代码的长度并没有变化，因为我们只是更改了一个用于描述输出尺寸的变量。然而，生成器的可能性空间规模现在缩小了，因为能够用 6x6 像素表示的花朵比用 12x12 像素表示的花朵少。图2 在图 1的同一张图表上展示了这两个生成器。请注意， F_6 在 x 轴上向左移动了，表示可能性空间更小；但它在 y 轴上的位置不变，因为代码长度没有变化。

由此可见，这一变化的结果之一是，生成器能够生成的最复杂作品也随之简化。总体而言，我们缩小了生成器输出的规模并降低了其复杂度，却没有减少其中编码的知识量。这使生成器聚焦于一组更小的输出。请注意，这不一定会让生成器变得“更好”，但它确实让生成器的输出更容易理解和描述，因为其中包含的信息和变化更少。在本例中，《茶园》的设计师更倾向于在最终设计中用 F_{12} 替换 F_6，因为生成精灵图的复杂度降低，更符合游戏低分辨率像素美术的美学风格。图 4 对比展示了 F_{12} 与 F_6的输出。

5.2 《我的世界》

《我的世界》是一款 3D 生存建造游戏，也是最著名的以程序化生成作为核心特色的游戏之一。《我的世界》的背景设定在程序化生成的世界中，这些世界远比任何玩家能够探索的范围大得多。游戏使用的算法经过多年的精心迭代，创造出壮观、有趣而美丽的

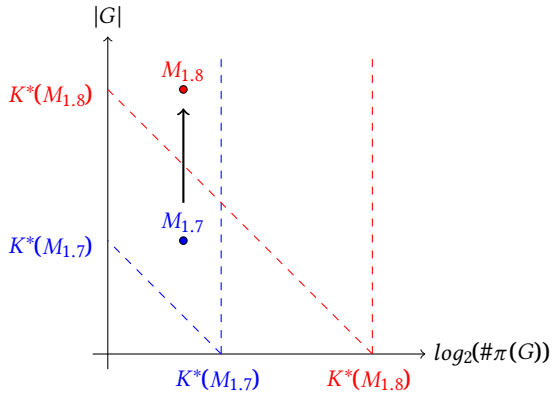

图 5. 展示《我的世界》世界生成器两个版本 $M_{1.7}$ 和 $M_{1.8}$ 之间变化的图表。

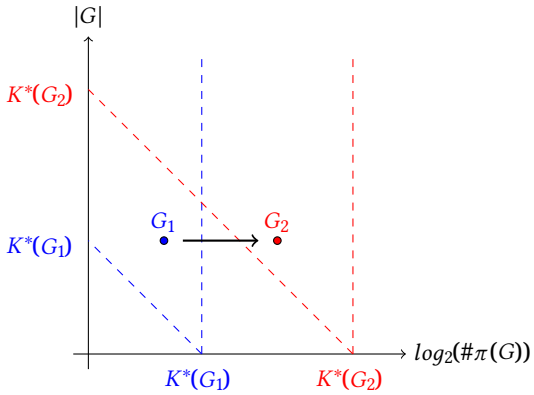

图 6. 展示假设生成器发生变化并最终生成“燕麦粥”的示意图。

世界，这些世界也发挥着重要的游戏功能，例如带来挑战、启发和惊喜。《我的世界》世界生成器的历史可从其补丁说明和社区记录中窥见 [5]。我们在此将生成器的每个版本称为 M_n，其中 n 是与之对应的《我的世界》主版本号。

在 $M_{1.7}$ 与 $M_{1.8}$之间，《我的世界》的设计者在其世界生成器中加入了村庄。在生成过程中，生成器会标记一块区域以放置村庄，然后使用房屋、农场和其他建筑的模板来建造村庄。图 5 展示了 $M_{1.7}$ 与 $M_{1.8}$ 在与图 1相同的图表上。

由于村庄的大部分方块都放置在原本会是空旷的空间中，我们认为这一特性表明《我的世界》的设计者希望提升其生成世界的复杂度（$K^*(G)$）。由于《我的世界》的两个版本都使用 64 位种子，最多可生成 2_{64} 个世界，因此这两个生成器的规模相同（$log_2(\#\pi(G))$），在 x 轴上的位置也相同。该定理预测，如果 $M_{1.7}$ 与 $M_{1.8}$ 之间的复杂度增幅足够大，《我的世界》的源代码长度也必须随之增加。

在这种可能的情况下，《我的世界》的设计师会选择提高其世界的复杂度，同时将更多设计师知识编码到世界生成器中。要提高其制品的复杂度，就必须在生成器上投入技能和资源；这一要求得到了相关 PCG 设计竞赛存在的支持，例如专门为《我的世界》生成村庄的 GDMC 人工智能聚落生成挑战赛。

5.3 燕麦粥的成本

在本节中2.3 我们描述了一万碗燕麦粥问题：系统生成了大量内容，但内容的质量和多样性都如此之低，以至于数量非但没有带来益处，反而成了问题。在本例中，我们将探讨在试图设计一个更复杂的生成器时，这种情况如何会在无意中发生。

假设我们已经开发出一个生成器，G_1 如图 6所示，并且希望使其输出更加复杂——也就是说，希望提高模式密度，$K^*(G_1)$。提高 K^* 最终将要求我们改变程序的长度（$|G|$）或可能性空间的大小（$log_2(\#\pi(G))$），因为我们的定理保证，对于任何理想生成器而言，不等式$|G| + log_2(\#\pi(G)) \geq K^*(G)$ 都成立 G。作为开发者，这为我们提供了两种解决方案。第一种是增加程序的长度，向生成器中加入更多编码知识。然而，这种方案既成本高昂又耗时。第二种方案是扩大可能性空间的大小，并扩展

生成器，例如随机组合我们的制品的子组件。这个方案成本低廉，因此颇具吸引力，并会产生类似 G_2 图 6中的生成器。可是，以这种方式线性增加 K^*，我们却在 指数级地 扩大可能性空间的大小，却没有添加任何新的编码知识来控制或塑造输出。这极有可能导致产生大量嘈杂、感知上相似且平庸的内容，也就是所谓的 燕麦粥。我们认为，燕麦粥现象的一个常见原因是燕麦粥很便宜，因为要提高生成器的模式密度，并不需要付出高昂代价来编码知识。

6 未来工作

本文初步探讨了将复杂性理论中的观点与生成系统联系起来的方法。这项工作可以从许多方面拓展，以涵盖更多主题或探索新的应用。例如，柯尔莫哥洛夫复杂度并非为神经网络而设计，但蓬勃发展的基于机器学习的程序化内容生成领域使其成为该领域值得关注的重要方向 [10]。我们也计划在未来研究如何利用柯尔莫哥洛夫复杂度来表达对 PCGML 系统的约束。

7 结论

本文介绍了信息论中的柯尔莫哥洛夫复杂度概念，并将其与程序化内容生成的研究联系起来。具体而言，我们证明了生成器源代码的长度、其可能性空间的大小，以及生成器所能生成内容的最高柯尔莫哥洛夫复杂度三者之间的关系。随后我们指出，这体现了程序化生成实践中一种广为人知的权衡：程序化生成器输出的规模、空间的稠密或嘈杂程度，以及其生成算法的精细程度之间存在取舍。最后，我们通过若干现实案例展示了这一观点如何应用于生成系统。

致谢

我们感谢 Azalea Raad 对我们证明的反馈，也感谢审稿人抽出时间提供建设性意见。

变换非理想生成器

设 G 是一个非理想生成器，它可以终止，但没有固定输入长度（不过它接受的输入集合有限，且不接受无限输入），并且不是单射。设 I 为 G 的定义域（即其输入集合），并设 m 为最长 $i \in I$ 的长度。

定义 enc_m 为一个函数，其输入为二进制字符串 $i \in I$，并返回以二进制字符串表示的 $|i|$，记作 b_i；如果 $|b_i|$ 小于 $\lceil log_2(m) \rceil$，则在其前面补零，再与 i 拼接；如果 $|i|$ 小于 m，则在其前面补零。设 $I'' = \{enc_m(i) : i \in I\}$；注意，$enc_m(i)$ 是一个单射函数，因为它会将每个 $i \in I$ 变换为一个唯一的字符串 $i'' \in I''$，其长度为 $m + \lceil log_2(m) \rceil$。

设 G'' 为定义域为 I'' 的生成器，使得对于所有 $i \in I''$，均有 $G''(i) = G(enc_m^{-1}(i))$（由于 enc_m 是单射，因此 enc_m^{-1} 定义良好）。于是，生成器 G'' 满足固定输入长度性质，因为 I'' 中的每个字符串长度都固定为 $m + \lceil log_2(m) \rceil$，且 G'' 的定义域为 I''。

设 G' 是一个定义域为 I'' 的生成器，使得对于所有 $i \in I''$，$G'(i) = G''(i) + i$。也就是说，给定输入 i 时，G' 的输出是 G'' 给定输入 i 时的输出与 i 本身的拼接。注意，G' 是单射：对于任意两个输入 $i, j \in I''$，如果 $i \neq j$，那么 $G'(i) = G''(i) + i$ 且 $G'(j) = G''(j) + j$，因此 $G'(i) \neq G'(j)$。此外，由于 G' 的定义域是 I''，它也满足固定输入长度。因此，G' 是理想生成器。

参考文献

[1] David Braben 和 Ian Bell。1984。《Elite》。[2] Michael Cook、Jeremy Gow、Gillian Smith 和 Simon Colton。2022。Danesh：用于理解程序化内容生成器的交互式工具。*IEEE* 游戏汇刊 14, 3 (2022), 329–338。[3] 安德烈·N·柯尔莫哥洛夫。1998。关于随机数表（转载自《Sankhya：印度统计学杂志》A辑，第25卷第4部分，1963年）。理论计算机科学 207, 2 (1998), 387–395。[4] Julian Mariño、Willian Reis 和 LeviLelis。2021。程序化生成的马里奥关卡评估指标的实证评估。*AAAI* 人工智能与互动数字娱乐会议论文集 (2021)。[5] 多位作者。2022。《我的世界》世界生成历史。https://tinyurl.com/mcwiki22。[6] Emily Short。2015。《The Annals of the Parrigues》。[7] Gillian Smith 和 Jim Whitehead。2010。分析关卡生成器的表达范围。载于*2010*年游戏程序化内容生成研讨会论文集。[8] Adam Summerville。2018。拓展表达范围：程序化内容生成的评估方法。载于第十四届 *AAAI* 人工智能与互动数字娱乐会议论文集。116–122。[9] Adam Summerville、Julian R. H. Mariño、Sam Snodgrass、Santiago Ontañón 和 Levi H. S. Lelis。2017。理解马里奥：平台游戏设计指标评估。载于第*12*届国际数字游戏基础会议论文集。ACM。[10] Adam Summerville、Sam Snodgrass、Matthew Guzdial、Christoffer Holmgård、Amy K. Hoover、Aaron Isaksen、Andy Nealen 和 Julian Togelius。2018。基于机器学习的程序化内容生成（PCGML）。*IEEE* 游戏汇刊 10, 3 (2018年9月), 257–270。