

---

## 第6章- GVGA I 中的程序内容生成

艾哈迈德·哈利法和朱利安·托格留斯

程序内容生成(Procedural Content Generation, PCG)是指使用计算机程序/算法自动生成游戏内容[16]。这个内容可以是游戏中的任何内容,比如纹理[20]、关卡[12、8]、规则[7]等。PCG在研究中可能有点新,但它从电脑游戏开始就存在了。它被用作游戏设计元素,允许重玩性和新场景,如《Rogue》(Glenn Wichman, 1980)。此外,它还被用于允许新类型的游戏使用小足迹,如《精英》(David Braben和Ian Bell, 1984)具有巨大的内容。尽管越来越多的人开始在游戏中使用PCG并发现新技术,但却没有人能够解决普遍性问题。业界使用的大部分技术都是建设性的[15],依赖于大量基于游戏知识的hack和技巧。能够适应不同游戏并在不同游戏之间工作的算法仍然是PCG领域的梦想。这就是在GVGA I 中拥有PCG轨道的主要动机。我们想给人们一个框架,帮助他们研究并找到更通用的方法,可以生成新的内容,与当前游戏的少量信息。

GVGA I 框架有几个决定游戏的不同方面:关卡、交互集、终止条件和精灵集(图形表示和类型)。我们决定从解决关卡生成问题开始,因为关卡生成是人们自20世纪80年代初开始就在解决的最古老且具有挑战性的问题之一。后来,我们为规则生成设计了一个新的轨道,在这个轨道中,用户必须同时生成交互集和终止条件,同时考虑到其他方面。这两个轨道的核心挑战之一是如何定义一个很好的方式来比较不同的生成技术。对于比赛,我们依靠人类来为我们提供这些数据,基于比较两个来自不同生成器的生成内容。人类对来自不同 games 的每个生成器的例子进行了多次比较,以确保算法可以在不同 games 之间进行适应,也可以为同一款游戏生成不同的关卡。这种技术适用于竞赛,但它并不能帮助用户理解如何增强他们的生成器,因为没有具体的规则集来定义什么是一个好的关卡或游戏。在本章中,我们将讨论这两个不同的轨道:关卡(第1节)和规则生成(第2节)。对于每一个,我们将dis-

咒骂他们的界面，提供的样例生成器，以及使用的最新技术和生成器。

## 1 GVGAI中的关卡生成

关卡生成赛道在2016年作为新的比赛赛道[8]引入。它被认为是GVGAI框架的第一个内容生成轨道。比赛的重点是创建提供游戏描述的可玩关卡。在这条赛道上，每个参与者提交一个关卡生成器，它在固定的时间内生成一个关卡。框架为生成器提供游戏精灵、交互集、终止条件和关卡映射，作为回报，生成器生成一个2D角色矩阵，其中每个角色代表该位置的游戏精灵。框架还允许生成器在需要时提供自己的关卡映射。

公共抽象类AbstractLevelGenerator

```
公共抽象String generatelevel(GameDescription game, ElapsedCpuTimer elapsedTimer);
```

```
public HashMap<Character, ArrayList<String>> getLevelMappingO
```

```
返回null;
```

图1:AbstractLevelGenerator类函数。

游戏信息通过GameDescription对象和ElapsedCpuTime对象提供给生成器。图1显示了参与者应该扩展的AbstractLevelGenerator类，以创建他们的关卡生成器。GameDescription对象是一个结构化的数据，它以一种更有组织的方式提供对游戏信息的访问，而ElapsedCpuTime对象为用户提供了生成器完成的剩余时间。当运行竞争时，ElapsedCpuTime定时器给每个生成器5个小时来完成它的工作。

```

1 BasicGame
2   SpriteSet
3     floor > Immovable randomtiling=0.9 img=oryx/floor3 hidden=True
4     goal  > Door img=oryx/doorclosed1
5     key   > Immovable img=oryx/key2
6     sword > OrientedFlicker limit=5 singleton=True img=oryx/slash1
7     movable >
8       avatar > ShootAvatar stype=sword frameRate=8
9       nokey   > img=oryx/swordman1
10      withkey  > img=oryx/swordmankey1
11      enemy    > RandomNPC
12      monsterQuick > cooldown=2 cons=6 img=oryx/bat1
13      monsterNormal > cooldown=4 cons=8 img=oryx/spider2
14      monsterSlow  > cooldown=8 cons=12 img=oryx/scorpion1
15      wall         > Immovable autotiling=true img=oryx/wall3
16
17   InteractionSet
18     movable wall > stepBack
19     nokey goal > stepBack
20     goal withkey > killSprite scoreChange=1
21     enemy sword > killSprite scoreChange=2
22     enemy enemy > stepBack
23     avatar enemy > killSprite scoreChange=-1
24     nokey key > transformTo stype=withkey scoreChange=1 killSecond=
25     True
26
27   TerminationSet
28     SpriteCounter stype=goal win=True
29     SpriteCounter stype=avatar win=False

```

Listing 1: VGDL Definition of the game *Zelda*.

除了这个对象，框架还提供了一个名为GameAnalyzer的辅助类，它分析GameDescription对象，并提供生成器可以使用的附加信息。GameAnalyzer将游戏精灵划分为5个不同的事件类型。

- 角色精灵:由玩家控制的精灵。
- 固体精灵:阻止玩家移动且没有其他互动的静态精灵。
- 有害精灵:能够杀死玩家的精灵或生成能够杀死玩家的另一个精灵。
- 可收集的精灵:玩家可以在与之碰撞时摧毁的精灵，为玩家提供分数。
- 其他精灵:不属于上述类别的任何其他精灵。

GameAnalyzer还为衍生精灵和目标精灵提供了两个额外的数组。它还根据次数为每个精灵提供一个优先级值

精灵的名字	精灵的形象	雪碧 类型	派生 雪碧	一级雪碧	价值
地板上		其他	假	假	0
目标		其他	假	真正的	3
关键		具有收藏价值的	假	假	1
剑		其他	真正的	假	1
nokey		《阿凡达》	假	真正的	4
本月中旬		《阿凡达》	假	真正的	2
monsterQuick		有害的	假	假	4
monsterNormal		有害的	假	假	4
monsterSlow		有害的	假	假	4
墙		固体	假	假	1

表1:《塞尔达传说》的GameAnalyzer数据。

每个精灵要么出现在交互中，要么出现在终止条件中。衍生精灵数组包含所有可以由另一个精灵生成的精灵，而目标精灵数组包含在游戏终止条件下出现的所有精灵。表1显示了清单1中需要的VGDL游戏塞尔达的GameAnalyzer输出。Nokey、withkey和goal是在终止条件中出现的唯一的目标精灵。剑是该游戏中唯一的衍生精灵，因为它只在化身攻击时出现(化身生成一把剑进行攻击)。精灵的类型是根据这些对象之间的相互作用以及它们的精灵类来分配的。例如:withkey和nokey是化身精灵，因为它们属于精灵职业ShootAvatar，而monsterQuick、monsterNormal和monsterSlow是有害的精灵，因为它们可以在碰撞时杀死化身精灵，如交互设定中的defined。

在以下小节中，我们描述了与GVGAI框架和竞赛一起提供的示例生成器，以及在IJCAI 2016, CIG 2017和GECCO 2018中提交竞赛或在2018年或之前发布的其他生成器。

## 1.1 样本发电机

本节详细讨论GVGAI框架提供的三个示例生成器。除了讨论它们如何工作和功能之外，我们还在最后详细说明了它们相对于彼此的优势，并展示了验证我们主张的先前的用户研究。

这是GVGAI框架中已知的最简单的生成器。该算法首先创建一个随机大小的关卡，与游戏精灵类型的数量成正比。然后，它会在关卡周围添加一个实线边框，以确保所有游戏精灵都留在里面。最后，它以10%的几率遍历每个贴图，添加一个随机选择的精灵，并确保每个游戏精灵至少出现一次，关卡中只有一个头像。

这是一个简单的生成器，与使用GameAnalyzer数据来提高生成关卡质量的框架一起提供。图2总结了样例构造生成器的核心步骤。生成器由四个步骤组成，加上预处理和后处理。

1. (预处理)计算覆盖百分比:这个预处理步骤帮助生成器定义地图大小和将被精灵覆盖的瓷砖的百分比。地图的大小取决于游戏中游戏精灵的数量，而覆盖百分比则与每个精灵的优先级值成正比。
2. 构建关卡布局:这是生成中的第一步。算法首先在关卡周围构建一个固体边界，然后它在关卡内部添加更多固体对象，这些对象相互连接，不阻塞任何区域。
3. 添加一个化身精灵:生成器在关卡中任意随机的空白空间中添加一个单独的化身精灵。
4. 添加有害精灵:生成器将有害精灵添加到与角色距离成比例的地图中，以确保玩家不会在游戏一开始就死亡。
5. 添加可收集精灵和其他精灵:生成器在地图的随机空位置添加可收集精灵和其他精灵。
6. (后处理)固定目标精灵:生成器确保在终止条件下目标精灵的数量大于指定的所需精灵数量。如果不是这样，生成器就会添加更多的目标精灵，直到发生这种情况。

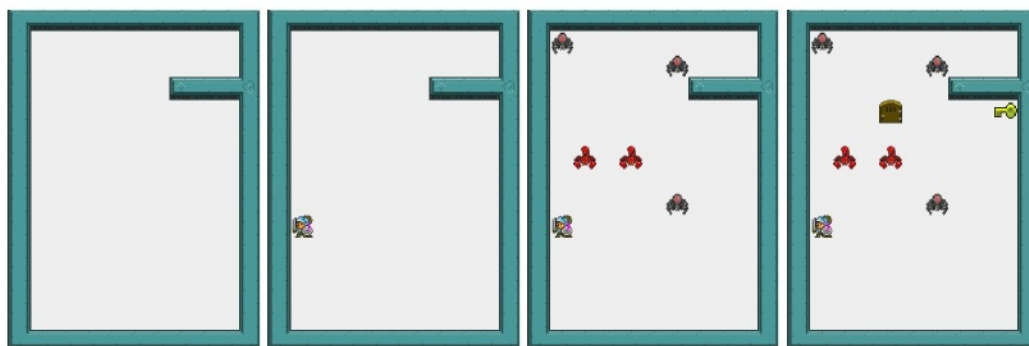


图2:VGDG《塞尔达传说》构造生成器中应用的步骤。左上角:构建关卡布局。右上角:添加一个头像精灵。左下:添加有害精灵。右下:添加收集物和其他精灵。

样本遗传生成器是一种基于可行非可行2种群遗传算法(FI2Pop)的基于搜索的关卡生成器[9]。FI2Pop是一种遗传算法，它跟踪两个种群:可行种群和不可行种群。不可行种群包含不满足问题约束的染色体。另一方面，可行种群试图为问题提高整体性能。在生成过程中的任何一点，如果染色体不满足问题约束，它就会转移到不可行种群，反之亦然。初始种群使用构造性生成器进行填充，其中每个染色体以二维瓦片数组的形式表示一个生成的水平。生成器围绕任何随机瓦片使用一个点交叉和三个变异算子。

- Create:在一个空的tile位置创建一个随机的sprite。
- 摧毁:清除随机贴图位置的所有精灵
- 交换:在关卡中随机交换两个瓦片。

我们使用三种不同的游戏控制器来评估生成的关卡:OLETS的修改版本、OneStepLook Ahead和donnothing。OLETS是2014年GVGAI比赛单人规划赛道的获胜者[14]。该算法基于开环Expectimax算法(更多细节请查看第4章)。该算法通过引入动作重复和每当智能体改变方向时的NIL动作，被modified玩得更像人类。这个modified阳离子被添加到sequence生成的关卡中，不需要超级人类的反应时间来解决它们。OneStepLookAhead只是一个简单的贪婪算法，它为立即的后续移动选择最佳动作。donnothing是一个简单的算法，它不执行任何操作。

这些代理被用作fitness计算和约束求解的一部分。将可行种群fitness分为两部分。

- 相对算法性能:算法的第一部分计算修改后的OLETS与OneStepLookAhead之间的性能差异。这是基于Nielsen等人的[13], 假设一个设计好的关卡/游戏在好的玩法和坏的玩法之间有一个高性能的difference差异。
- 独特的互动:fitness的第二部分计算score在游戏中由于玩家或由玩家衍生的对象而产生的独特互动的数量。我们假设一个好的生成关卡应该有足够的互动, 可以被玩家score红色, 以保持游戏的趣味性。

对于不可行群体, 算法尝试满足7个different约束。这些约束是基于我们对GVGA的理解和我们对优秀关卡的直觉而设计的。

- 角色数量:每个关卡只能有一个角色, 不多也不少。这个限制保证了游戏开始时只有一个可控制的角色。
- 精灵数量:每个关卡必须至少有一个来自每个未衍生的精灵的精灵。这一约束确保了关卡没有遗漏任何可能需要获胜的必要精灵。
- 目标数量:每个关卡的目标精灵数量必须高于终止条件中指定的数量red。这一约束确保了在游戏开始时, 关卡不会自动以胜利结束。
- 覆盖百分比:贴图只能覆盖每个关卡的5%到30%之间。这个约束确保关卡既不为空也不挤满different精灵。
- 解长度:关卡必须在200步以内解决, 以确保关卡不琐碎。
- 胜利:关卡必须能够通过修改后的OLETS来赢得, 以确保玩家也能赢得它。
- 死亡:当使用donothing玩家时, 角色不能在40步内死亡, 以确保玩家不会在游戏的前几秒内死亡。

试点研究和讨论在之前的作品[8]中, 通过使用25名人类玩家的试点研究, 对样本生成器进行了相互比较。研究对象是三款不同的VGD游戏。

青蛙:是《Frogger》(Konami, 1981)的VGDL移植。游戏的目的是帮助青蛙达到目标,不被车撞,不被水淹死。

- 《吃豆人》:是《吃豆人》(Namco, 1980)的VGDL移植版本。游戏的目标是收集屏幕上的所有小球,不被追逐的幽灵抓到。

《塞尔达传说》:是《塞尔达传说》地下城系统的VGDL移植版本。游戏的目标是在不被敌人杀死的情况下,收集一把钥匙,到达一扇门。玩家可以用自己的剑杀死这些敌人,获得额外的分数。

在这项试点研究中,我们通过给每个算法5小时的生成时间上限,在每个游戏中生成3个关卡。每个用户都面临两个随机选择的生成关卡,他们必须决定是第1个关卡比第二个好,第二个关卡比第1个好,两者都一样好,还是两者都一样差。

	首选	非优先考虑	总计	二项假定值
搜索型vs建设性	23	12	35	0.0447
基于搜索vs随机	21	10	31	0.0354
建构vs随机	17	24	41	0.8945

表2:玩家对每个生成器的偏好在三场游戏中聚合。

我们假设基于搜索的关卡比构造性关卡要好,而构造性关卡比随机生成的关卡要好。表2显示了这三款游戏的研究结果。在那张表中,我们只保留了两种水平中任何一种比另一种好时的结果,删除了所有两种水平都一样好或一样差的数据。从表中我们可以确定基于搜索的生成关卡比构造和随机都要好,但让人感到惊讶的是,构造生成器和随机生成器在同一层上。从进一步的结果分析来看,我们认为结果是构造生成器并不能确保每个关卡不同精灵类型中至少有一个对象,这导致了一些构造关卡与随机生成关卡相比是无法解决的。

## 1.2 竞争和其他生成器

我们可以根据关卡生成器的核心技术将其分为三大类:构造方法、基于搜索的方法和基于约束的方法。样本随机生成器和样本构造生成器都是构造方法,而样本遗传生成器是一种基于搜索的方法。在



接下来，我们将深入讨论这些算法及其核心技术。

**构造方法**构造方法使用基于game spec知识，通过在关卡中放置游戏精灵直接生成关卡，例如不要将玩家放置得太靠近敌人，不要添加隔离地图区域的固体精灵等。这些生成器在生成后不会检查可玩性，因为它们应该以一种避免这些问题的方式设计，并确保生成的内容始终是可玩的。

**Easablade建设性发电机**:IJCAI 2016级发电机大赛冠军。这个生成器在多个层上使用细胞自动机[6]来生成关卡。第1层负责设计地图布局，其次是出口精灵和化身精灵，然后是目标精灵、有害精灵等。

**N-Gram构造生成器**:已提交CIG 2017关卡生成大赛。它使用n-gram模型来生成关卡。生成器使用录制的游戏通关来生成关卡。该算法为每个不同的类型的互动提供案例，例如:攻击涉及在关卡的某些地方添加敌人，围绕特定瓷砖行走涉及添加固体物体等。ngram模型用于指定这些规则，所以系统不是对单个动作做出反应，而是对n个动作序列做出反应。在生成过程中，算法也会跟踪所有放置的精灵，以确保它不会过度填充关卡。然后，在关卡底部中心的头像精灵被添加。

**Beaupre的构造模式生成器**:是为Beaupre等人[1]分析在GVGAI框架中自动生成关卡时使用设计模式的效果而设计的构造算法。在他们的工作中，他们分析了来自GVGAI框架的97款不同游戏，在将关卡转换为使用GameAnalyzer的精灵类型而不是实际精灵后，他们在所有关卡上使用3 × 3滑动窗口。他们构建了一个包含所有这些different模式的字典(他们发现了12,941个独特的模式)，并根据混合在一起的different对象的类型对其进行分类。有出现在关卡边界上的边框图案，有包含头像精灵的头像图案，还有其他图案。算法首先检查游戏是否有实体精灵。如果是这样的话，它就使用边框图案来定位边框区域。关卡的其余部分是根据它们的分布从其余的图案中随机选择的，同时确保关卡中只有头像，并且关卡仍然是完全连接的。最后，系统将通用模式转换

精灵类型转换为specific游戏精灵, 同时确保目标精灵的数量大于终止条件下的specific数字。

法兰克福学生建设性生成器:为GECCO 2018级别生成比赛而设计。这个生成器是更手工的生成器, 试图识别游戏的类型(赛车游戏, 捕捉游戏等), 并基于该类型的differences的水平大小, 并给每个游戏精灵一个优先位置相对于其他精灵和水平。

基于搜索的方法基于搜索的方法使用基于搜索的算法(如遗传算法)来创建可玩且比随机放置精灵更有趣的关卡。本节介绍了所有已知的基于搜索的生成器。

Amy12基因生成器:是建立在样本基因生成器之上的。该发电机已提交给IJCAI 2016竞赛。这个想法是为了生成具有一定悬念曲线的关卡。悬念是通过使用OLETS代理测量玩法过程中导致死亡的动作数量来计算的。该算法试图修改关卡, 得到一条悬念曲线, 其中三个峰值点的高度小于50%。这样做的好处是, 它可以确保生成的关卡是可以获胜的, 因为无法获胜的关卡将在悬念曲线中拥有更高的峰值。

Jnicho遗传生成器:与样本生成器[12]中使用的FI2Pop相比, 使用了标准的遗传算法。生成器将约束条件和相对算法性能结合在一个fitness函数中, 其中相对算法性能是在MCTS代理和OneStepLookA-head代理之间进行测量的。这些代理的得分值被归一化在0到1之间, 以确保相对的算法性能不会掩盖其余的约束。该发电机已提交给IJCAI 2016竞赛。

Number13基因生成器:是样本基因生成器的修改版本。该发电机已提交给IJCAI 2016竞赛。该生成器使用自适应方法进行交叉和变异率, 具有比modified OLETS表现更好的代理。它还允许可行染色体和不可行染色体之间的交叉, 这在样本生成器中是不允许的。

博普雷的进化模式生成器:使用与构造方法中描述的构造生成器相似的思想。这个生成器是与框架提供的样本遗传生成器的modified版本, 但染色体表示为图案的2D数组, 而不是tiles。在这种情况下, 他们使用他们的构造性方法来初始化初始种群。

Sharif的模式生成器:作为GECCO 2018竞赛的参与者提交。这个生成器类似于博普雷的进化模式生成器

使用identified模式来构建关卡。在Sharif等人的先前工作中[17]，他们确定了在生成过程中使用的一组23个不同的独特模式。这些图案被选中中具有与Dahlskog和Togelius在《超级马里奥兄弟》(任天堂，1985)[5]生成关卡的作品中所识别的图案相似的语义。

建筑师基因生成器:是GECCO 2018级别生成竞赛的获胜者。该算法建立在框架提供的样本遗传生成器上，它是唯一使用两点交叉和新的构造性初始化技术的difference。新的构造性技术类似于样本随机生成器，但有一些改进。它首先计算地图的大小，然后构建一个类似于样本构造技术中使用的关卡布局，然后在地图中添加一个头像。最后从所有可能的精灵中随机选择10%的地图。

Luukgvgai基因生成器:是另一个GECCO 2018提交，类似于另一个是使用锦标赛选择和两点交叉的样本基因生成器的改进版本。

Tc遗传生成器:是样本遗传生成器的改进版本，它使用8点交叉和锦标赛选择而不是排名选择。它还使用了级联fitness，其中相对算法性能的优先级高于独特的交互。该算法已提交给GECCO 2018 GVGAI级别生成竞赛。

基于约束的方法基于约束的方法使用约束求解器来生成关卡。生成器的想法是find所需的正确约束，以确保生成的内容是有针对性的体验。到目前为止，使用该技术仅为GVGAI开发了一个发电机。

ASP生成器:[11]使用答案集编程(Answer Set Programming, ASP)生成关卡。ASP程序使用进化策略进行进化，该策略使用了sampleMCTS代理和sampleRandom代理之间的相对算法性能。进化后的规则分为三种不同的类型。first类型是一套基本的规则，可以确保生成的关卡不复杂，比如确保每个贴图只有一个精灵。第二种类型更具有游戏特色，规则基础，例如识别当前游戏中的单例精灵。第三种类型关注的是可以为特定精灵类型产生的最大对象数量。

### 1.3 讨论

所呈现的生成器在生成关卡所需的时间和生成内容的特征上。建设性生成器在不保证生成的关卡是可打败的情况下，花费最少的时间来生成单个关卡。另一方面，基于搜索和基于约束的生成器都需要更长的时间，但却能够生成具有挑战性的可击败关卡，因为它们使用自动游戏代理作为fitness函数。基于约束的生成器只需要很长时间来生成一个ASP生成器，它可以用来生成许多不同的事件关卡，与建设性生成器一样快，而基于搜索的生成器需要很长时间来生成一组相似的关卡。

提出的一些算法已提交给IJCAI 2016 (Easablade、Amy12、Number13和Jnichogenerator)、CIG 2017 (N-Gram建设性生成器)和GECCO 2018 (Architect、FrankfurtStudents、Sharif、Luukvgai和Tcru生成器)的GVGAI级别生成轨道。这些生成器在比赛中相互进行了比较。

图3显示了IJCAI 2016竞赛的结果。

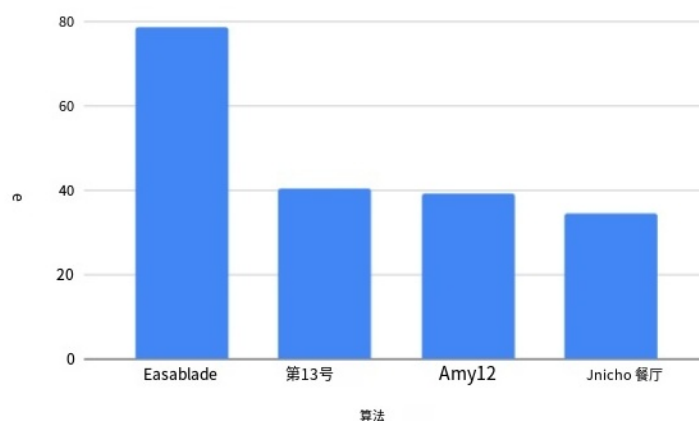


图3:IJCAI 2016水平生成轨迹结果。

我们针对4款GVGAI游戏测试了这些算法:

- 黄油bunnies:是一款关于在关卡中打开所有茧之前收集所有黄油bunnies的VGDL游戏。
- Freeway:是Freeway的VGDL端口(David Crane, 1981)。类似于青蛙，目标是到达道路的另一边，不被汽车辗过。的

Diifference是，这款游戏是一款基于分数的游戏，玩家需要多次达到目标才能获得更好的分数。

-Run:这是一款VGDL奔跑类游戏，玩家需要尝试着逃离来自身后的敌人并及时到达出口。

-雪人:在这个游戏中，目标是将雪人的碎片(腿、躯干和头)按照正确的顺序堆叠起来，创造一个雪人。

结果非常清楚，Easablade击败了其他三个生成器。一个可能的原因是，Easablade在生成的场景中生成了一些具有良好布局的精灵。另一方面，所有生成的关卡要么是好玩的(出口藏在墙后)，要么是容易打败的(出口就在玩家旁边)。

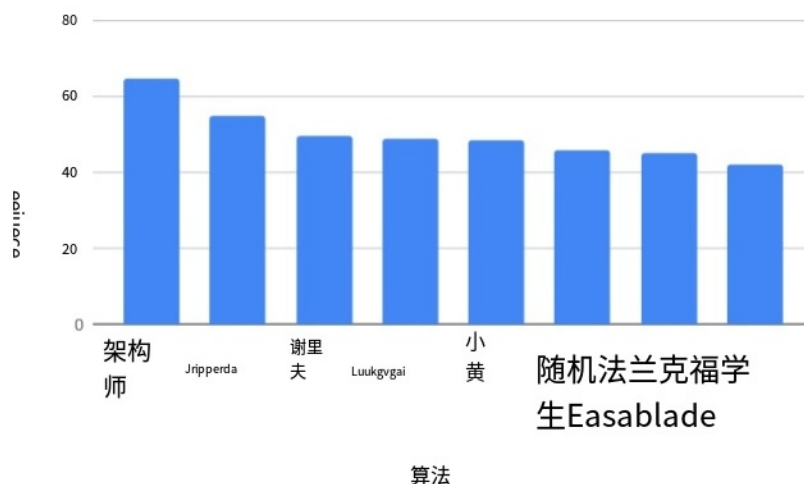


图4:GECCO 2018水平生成轨迹结果。

图4显示了GECCO 2018竞赛的结果。我们对从stepheston等人的作品[18]中挑选的三个GVGA游戏进行了算法测试，以找出最具歧视性的VGDL游戏。

-Chopper:是一款VGDL动作游戏，玩家需要收集子弹来杀死来袭的坦克而不被敌方坦克的子弹杀死。同时，玩家还要保护卫星不被摧毁。

-迷宫:是一款VGDL益智游戏，玩家试图通过改变颜色来通过彩色障碍，在不被杀死的情况下到达出口。

是一款VGDL解谜游戏，关于在不溺水的情况下到达出口。为了通过关卡，玩家需要推动一些药剂将水转化为floor，这样玩家才能通过它。

与Easablade相比，Architect以微弱的优势赢得了比赛。获奖背后的原因尚不清楚，因为其他一些发电机也有类似的外观关卡。令人震惊的是，与随机生成器相比，Easablade获得了最差的评级。我们认为游戏a的选择影响了比赛的结果。例如，像《Watergame》这样的益智游戏只有在可解决的情况下才具有可玩性，拥有较大的关卡和稀疏的对象将违背生成器的兴趣，而像《高速公路》这样的游戏可能会更好。

## 2 GVGAI中的规则生成

规则生成赛道于2017年作为新的比赛赛道[7]推出。它是GVGAI框架的第二代轨道。比赛的重点是创建一个新的可玩的游戏提供的游戏级别与所有的游戏精灵。在这个赛道上，每个参与者提交一个规则生成器，该生成器在固定的时间内生成交互集和终止条件。该框架为生成器提供了2D角色矩阵形式的游戏关卡，可以使用提供的关卡映射将其转换为相应的游戏精灵。作为回报，生成器生成两个包含游戏交互集和终止条件的字符串数组。

```
public abstract class AbstractRuleGenerator {

    public abstract String[][] generateRules(SLDescription sl, ElapsedCpuTimer time);

    public HashMap<String, ArrayList<String>> getSpriteSetStructure(){
        return null;
    }
}
```

图5:AbstractRuleGenerator类函数。

游戏关卡和游戏精灵是通过SLDescription对象提供的。图5显示了用户需要扩展的AbstractRuleGenerator类来构建自己的规则生成器。用户必须实现generateRules函数，并返回包含游戏交互集和终止的两个数组

条件提供了SLDescription对象和ElapsedCpuTimer对象。SLDescription对象为生成器提供所有游戏精灵的列表。这些游戏精灵有名称、类型和相关的精灵。例如:在《塞尔达传说》中,如清单1所示,玩家角色名为avatar,类型为ShootAvatar(这意味着它可以向所有四个方向射击),并将剑作为相关精灵(因为它可以向四个方向中的任何一个方向射击)。除了游戏精灵,SLDescription对象还提供了一个2D矩阵,代表游戏关卡,其中每个单元格代表该单元格中的精灵。在运行竞赛时,ElapsedCpuTimer为生成器提供了5个小时来完成生成。

用户还可以覆盖一个名为getSpriteSetStructure的可选函数,该函数返回一个字符串和字符串数组之间的hashmap。hashmap代表了规则生成过程中需要的子画面层次结构。举个例子:在《塞尔达传说》中,用户可以生成三条规则,当角色击中任何怪物时杀死角色quick、monsterNormal和monsterSlow精灵,或者可以生成一条规则,当角色击中有害的精灵时杀死角色,然后在精灵结构中define有害的精灵作为这三个怪物。



图6:《塞尔达传说》提供的关卡

该框架还为用户提供了一个LevelAnalyzer对象,用于分析所提供的关卡,并允许用户询问覆盖地图特定百分比和/或特定类型的精灵。例如:生成器可以通过要求LevelAnalyzer获得覆盖100%关卡的不可移动精灵来获得背景精灵。这些信息可以用来将游戏精灵分类到不同的类别。表3显示了一些可以从图6中呈现的塞尔达游戏中识别出来的示例类。这些类别是基于gen-的defined

一般的游戏知识。例如:《马里奥》中的硬币等得分对象覆盖了小比例的关卡。




类类型	精灵的形象	雪碧类型	阈值	周围的水平
背景		不动的	$\geq 100\%$	假
墙		不动的	$< 50\%$	真正的
分数/飙升		不动的	$< 10\%$	假

表3:《塞尔达传说》的LevelAnalyzer数据

我们运行了两次规则生成轨迹:在CIG 2017和GECCO 2018。在这两次比赛中,我们都没有收到任何参赛作品,因为赛道上没有广告,而且问题更难解决(为游戏生成规则比仅仅制作一个关卡更难)。在接下来的小节中,我们将讨论比赛中提供的样本生成器,以及我们在文献中找到的唯一的另一个生成器,它是在赛道存在之前创建的。

## 2.1 样本发电机

与关卡生成轨迹类似,规则生成轨迹带有三个different生成器:随机生成器、构造生成器和基于搜索的生成器。这些生成器增加了生成内容的复杂性和质量。

样本随机生成器样本随机生成器是所有生成器中最简单的。生成器只是选择任何随机交互和终止条件,将编译成没有任何错误的VGDL游戏。生成器按照以下步骤生成随机VGDL游戏。

1. 随机选择一些互动。
2. 通过重复以下步骤生成随机的交互量,直到达到所选择的交互量,同时确保它编译没有错误:
  - (a)随机选择两个精灵。
  - (b)随机选取一个scoreChange值。
  - (c)随机选择一个交互。
3. 生成两个输赢的终端条件。



获胜:要么是一个随机时间的暂停,要么是某个精灵计数达到零。

-失败:如果角色死亡,游戏就失败了。

样本构造生成器构造生成器比样本生成器更复杂。它使用基于模板的生成来生成一个可玩的好游戏。生成器有一个基于游戏知识设计的游戏模板。例如:如果有一个非可玩角色(NPC)在追逐某个物体,那么它很有可能在碰撞时杀死它。生成器利用SLDescription对象中的精灵类型和LevelAnalyzer对象中的精灵类模板。以下是示例生成器所采取的步骤。

1. 获取资源互动:为所有资源精灵添加了collectResource交互。
2. 获得分数和刺钉互动:每个物品有50%的几率是可收集的或有害的。可收集的精灵会在与角色碰撞时被杀死,并给予1分得分。Spike精灵与角色碰撞时击杀角色。
3. 获取NPC互动:不同类型的NPC会添加不同的互动。通常它们要么被化身收集到1分,要么在碰撞时杀死玩家。有些NPC可能会生成其他精灵,这些精灵可能是可收集的,也可能是致命的。
4. 获取刷出器互动:与NPC类似,刷出器精灵决定生成的精灵是杀死角色还是收集点数。
5. 获得传送门互动:如果传送门是门的类型,角色会在碰撞时摧毁它。否则,化身被移动到目的地传送门。
6. 获得可移动的互动:生成器随机决定这些可移动的物体是有害的还是可收集的精灵。
7. 获取墙壁互动:生成器决定墙壁精灵是在墙壁上还是在普通墙壁上。火墙会在碰撞时杀死任何可移动的精灵,而普通墙则会阻止任何可移动的精灵。
8. 获得角色互动:此步骤仅在生成有害精灵且角色可以射击时才会发生。生成器增加了子弹和有害精灵之间的互动,在碰撞时杀死两者。
9. 获取终止条件:

获胜:生成器选择一个随机的获胜条件,如角色到达一扇门,所有有害的精灵都死了,所有可收集的精灵都收集了,或者时间耗尽。

失败:当角色死亡时,游戏就失败了。

样本遗传生成器类似于关卡生成轨迹，基于搜索的算法使用FI2Pop算法[9]来生成交互集和终止条件。不可行的染色体试图通过满足这三个约束条件而变得可行。

- 相互作用集和终止条件编译时不会出错。
- 一个无所事事的代理会存活40帧。
- 坏帧的百分比小于30%，坏帧是指一个或多个精灵在屏幕边界之外的帧。

如果染色体满足 $\delta$ es这些约束条件，它就会被移动到可行种群。可行染色体试图改善它们的 $\delta$ 性。可行的 $\delta$ ness也由三部分组成。

提高三个不同代理(OLETS代理、MCTS代理和随机代理，按性能顺序排列)之间的相对算法性能[13]。

增加在游戏过程中使用的唯一规则的数量。游戏规则被添加使用，如果一个规则没有被任何代理使用，那么它违反了约束。

增加代理人赢得关卡所需的时间我们不希望在500帧(20秒)以内就能获胜的游戏。

我们使用了有90%机会进行交叉，10%机会进行变异的排名选择。我们使用单点交叉来切换交互集和终止条件。对于变异，我们使用了三个different算子。

- 插入:要么插入一个新的规则/终止条件，要么插入一个新的参数。
- delete:删除一个规则参数或删除整个规则/终止条件。
- Modify:修改其中一个规则参数或修改规则/终止条件本身。

我们用10条构造染色体、20条随机染色体和20条变异的构造染色体来初始化算法，a的种群大小为50。我们使用了2%的精英化，在两代之间保持最好的染色体。

我们将这三种生成器应用于根据Bontrager等人[2]的不同算法性能选择的三种不同的事件游戏(《外星人》、《卵石》和《Solarfox》)。我们的 $\delta$ rst研究是为了看到多样性

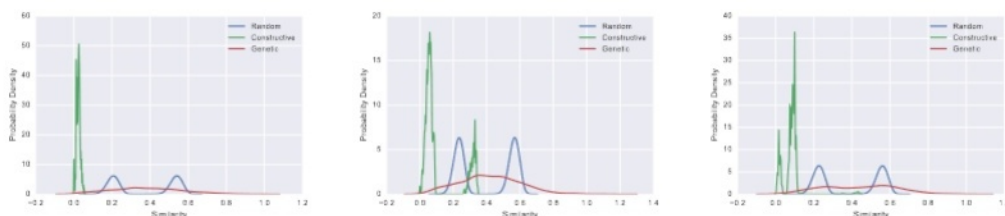


图7:生成的游戏之间的相似性度量的概率密度。这些图表从左到右分别代表外星人、卵石和Solarfox。0表示游戏是相同的，而1表示游戏完全不同different。

	外星人, 巨石, 太阳狐		
基因vs Rnd	2/8	7/7	11/15
遗传vs Const	0/14	8/14	6/18
Const vs Rnd	9/10	10/11	4/5

表4:我们的规则生成器之间的比较，其中Rnd是样本遗传生成器，是样本随机生成器，是样本构造生成器。rst值是用户偏爱rst游戏多于第二个的次数。第二个值是比较的总次数。

这些生成器在这三款游戏上的生成内容。图7显示了生成的游戏彼此之间相似程度的机率密度函数，其中0.0表示100%相似，1.0表示它们完全不同。这些分布是通过从样本随机生成器和样本构造代理生成1000个游戏和从样本遗传生成器生成350个游戏(由于时间限制)来计算的。从所有分布中，我们可以看到样本遗传生成器能够生成从彼此非常相似到完全different的游戏，而构造生成器非常局限于类似的游戏，这是意料之中的，因为我们正在使用一个具有少量参数需要更改的模板。

我们还对关卡生成器进行了类似的用户研究，以比较这三种生成器的偏好。用户面对的是来自相同生成器或different生成器的两个具有相同关卡布局的生成游戏，我们让他们选择他们认为最好的一个(rst或second)，两者都是同样优秀的，或者两者都不是优秀的。表4展示了用户研究的结果。如我们所料，除了基因生成器中的外星人外，构造生成器和遗传生成器大多优于随机生成器。外星人不同的原因是基因

生成器没有太多时间进化，所以为了满足约束条件，它生成了一个带有撤消所有的游戏，作为玩家和背景之间的交互，以满足糟糕的帧约束。撤消所有互动暂停整个游戏，不允许任何事情发生。另一种评论是，基因生成器并没有优先于建设性生成器。我们认为原因是构造模板设计得更好，用户从来没有注意到它们彼此相似。

## 2.2 其他发电机

据我们所知，只有Thorbjørn等人对规则生成进行了另一项工作。生成器类似于样本遗传生成器，因为样本生成器的想法就是基于那项工作。这两个生成器都使用不同算法性能之间的差异(不同算法的相对算法性能)作为评价函数。Difference是，这个生成器使用带有变异算子的进化策略来生成整个游戏，而不是交互集和终止条件。

## 2.3 讨论

规则生成竞赛已经进行了两年，但还没有人提交。我们认为这背后的原因可能是竞争比关卡生成更难(需要更多的计算)，并且通常与关卡生成同时运行，因此竞争对手要么选择参加关卡生成竞赛。我们认为在规则生成竞赛中存在着巨大的研究机会。一些研究人员致力于基于Ralph Koster Fun理论生成完整游戏[19][10]、通过修改游戏代码生成游戏[4]、基于单一作品[3]生成游戏等。

## 参考文献

1. S. Beaupre, T. Wiles, S. Briggs, and G. Smith, A Design Pattern Approach for Multi-Game Level Generation, in Artificial Intelligence and Interactive Digital Entertainment. AAAI, 2018.
2. P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, Matching Games and Algorithms for General Video Game Playing, in Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference, 2016.
3. M. Cook and S. Colton, A rogue dream: Automatically generating meaningful content for games, in Tenth Artificial Intelligence and Interactive Digital Entertainment Conference, 2014.
4. M. Cook, S. Colton, A. Raad, and J. Gow, Mechanic miner: Reaction-driven game mechanic discovery and level design, in European Conference on the Applications of Evolutionary Computation. Springer, 2013, pp. 284–293.
5. S. Dahlskog and J. Togelius, Patterns and procedural content generation: revisiting mario in world 1 level 1, in Proceedings of the First Workshop on Design Patterns in Games. ACM, 2012, p. 1.

6. L. Johnson, G. N. Yannakakis, and J. Togelius, [Cellular automata for real-time generation of infinite cave levels](#), in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010, p. 10.
7. A. Khalifa, M. C. Green, D. Pérez-Liebana, and J. Togelius, [General Video Game Rule Generation](#), in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2017.
8. A. Khalifa, D. Perez-Liebana, S. M. Lucas, and J. Togelius, [General video game level generation](#), in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM, 2016, pp. 253–259.
9. S. O. Kimbrough, G. J. Koehler, M. Lu, and D. H. Wood, [On a feasible–infeasible two-population \(2pop\) genetic algorithm for constrained optimization: Distance tracing and no free lunch](#), *European Journal of Operational Research*, vol. 190, no. 2, pp. 310–327, 2008.
10. R. Koster, *Theory of fun for game design*. O’Reilly Media, Inc., 2013.
11. X. Neufeld, S. Mostaghim, and D. Perez-Liebana, [Procedural level generation with answer set programming for general video game playing](#), in *Computer Science and Electronic Engineering Conference (CEECE)*, 2015 7th. IEEE, 2015, pp. 207–212.
12. J. Nichols, [The Use of Genetic Algorithms in Automatic Level Generation](#), Master’s thesis, University of Essex, 2016.
13. T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, [General video game evaluation using relative algorithm performance profiles](#), in *European Conference on the Applications of Evolutionary Computation*. Springer, 2015, pp. 369–380.
14. D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, [The 2014 general video game playing competition](#), *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.
15. N. Shaker, A. Liapis, J. Togelius, R. Lopes, and R. Bidarra, [Constructive generation methods for dungeons and levels](#), in *Procedural Content Generation in Games*. Springer, 2016, pp. 31–55.
16. N. Shaker, J. Togelius, and M. J. Nelson, *Procedural content generation in games*. Springer, 2016.
17. M. Sharif, A. Zafar, and U. Muhammad, [Design Patterns and General Video Game Level Generation](#), *Intl. Journal of Advanced Computer Science and Applications*, vol. 8, no. 9, pp. 393–398, 2017.
18. M. Stephenson, D. Anderson, A. Khalifa, J. Levine, J. Renz, J. Togelius, and C. Salge, [A continuous information gain measure to find the most discriminatory problems for ai benchmarking](#), *arXiv preprint arXiv:1809.02904*, 2018.
19. J. Togelius and J. Schmidhuber, [An Experiment in Automatic Game Design](#), in *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 2008, pp. 111–118.
20. G. Turk, [Generating textures on arbitrary surfaces using reaction-diffusion](#), in *ACM SIGGRAPH Computer Graphics*, vol. 25, no. 4. ACM, 1991, pp. 289–298.