第2章- VGDL和GVGAI框架

迭戈Perez-Liebana

1 介绍

本章描述了框架的核心:电子游戏描述语言(VGDL)和通用电子游戏AI(GVGAI)基准。GVGAI中的所有游戏都是通过VGDL来表达的，这是一种最初由Tom Schaul开发的基于文本的描述语言[6]。让游戏和关卡的描述独立于运行它们的引擎带来了一些优势。其中之一就是可以设计一种语言⊠t，它可以足够简洁和富有表现力，很多游戏都可以在其中开发。最生动的例子就是从一开始就用这种语言创造的200款游戏。这种分离的另一个有趣的优点是，不必每创建一个新游戏就重新编译框架，这有助于软件的可持续性。引擎可以保持编译后的二进制形式(或.jar⊠le)，只需要将游戏和关卡作为输入数据。

但可以说，拥有VGDL的主要好处之一是可以轻松定制游戏和关卡。它的简单性使研究人员和游戏设计师可以在几分钟内创造出当前游戏的变体。与其他游戏研究框架相比，调整VGDL游戏比修改街机学习环境(ALE)或Unity-ML游戏要容易得多——事实上，在某些情况下，这甚至是不可能的。这个功能不仅仅是简单的方便:它打开了di⊠不同的研究渠道。例如，它促进了游戏组合的创建，以避免在学习代理[4]中过度⊠tting，以及使自动游戏调整和平衡(见第7章)和游戏和关卡生成(第6章)的研究成为可能。

最后但并非最不重要的是，VGDL的多功能性简化了竞赛的组织，为其每个版本提供了新的挑战。多年来，竞争一直是游戏AI社区的一部分，通常一次只关注一款游戏。如上一章所述，一次性使用更多的游戏，可以让研究更多地关注方法的通用性，而不是每个游戏的具体情况⊠cs。然而，如果没有以相对快速和简单的方式生成中等复杂度游戏的能力，这并不容易实现。

本章的第2节详细描述了电子游戏描述语言，特别强调了它的语法和结构，也提供了单人和双人游戏的示例，使用网格和连续物理。第3节描述了解释VGDL的框架，并为不同类型的代理公开了一个API来玩可用的游戏。接下来，在第4节中，介绍了竞赛和多年来运行的di□erent版本。

2 电子游戏描述语言

VGDL是一种设计二维街机类游戏的语言。该语言是为围绕对象(精灵)创建游戏而设计的，其中一个对象代表玩家(化身)。这些对象具有特定的属性和行为，并且物理上位于一个2D矩形空间中。这些精灵能够成对地相互作用，这些作用的结果就形成了游戏的动态和终止条件。

VGDL将逻辑分成两个不同的组件:游戏和关卡描述，两者都需要纯文本格式。所有VGDL游戏描述符都有四个指令块。

-精灵集:本集de□nes游戏中可用的精灵。每个对象或精灵都是带有一个类和一组属性的de□ned。它们被组织在一个树中，所以一个子精灵继承了它祖先的属性。所有的类型和属性都在一个本体中被定义，所以每个精灵对应于一个游戏引擎(在本书中，这个引擎是GVGAI)中需要的类。

-交互集:这些指令de□ne当两个de□ned精灵相互碰撞时发生的事件。与前面的例子一样，可能的e□ects在一个本体中是de□ned，可以接受参数。交互按照它们在这个de□nition中出现的顺序被触发。

-终止设置:这个设置de□nes游戏结束的条件。这些条件按照本集合中指定的顺序进行检查□ed。

-关卡映射:这个集合建立了关卡中的角色□le和精灵集合中的精灵de□ned之间的关系。

清单1中的代码显示了该框架中一个游戏的完整描述:外星人。这款游戏是传统街机游戏《太空入侵者》的一个版本，在这款游戏中，玩家(位于屏幕底部)必须消灭所有来自上方的外星人。下面我们分析实现这款游戏的代码。

```
BasicGame方块大小=32
  SpriteSet
    背景>不可移动img=oryx/spacel隐藏=True
    基地          >不可移动img=羚羊/行星
    avatar > FlakAvatar style =sam img=oryx/太空船1导弹>导弹
    sam > orientation=UP singleton=True img=oryx/bullet1 bomb > orientation=DOWN
    speed=0.5 img=oryx/bullet2 alien
                  >轰炸机类型=炸弹prob=0.01冷却时间=3速度=0.8
      alienGreen > img=oryx/alien3
      alienBlue > img=oryx/alien1
    门户> invisible=True hidden=True
      portalSlow      >产卵点类型=alienBlue冷却时间=16 total=20
      portalFast >
  产卵点类型=alienGreen冷却时间=12 total=20 InteractionSet
    Avatar EOS >后退一步。
    外星EOS >转机
    导弹EOS >杀死精灵

    基地炸弹>杀死两者
    base sam > 杀死两者 scoreChange=1

    基地        外星人杀死雪碧
    avatar alien bomb > killSprite scoreChange=-1
    alien sam > killSprite scoreChange=2

  TerminationSet
    SpriteCounter                  药栓=《阿凡达》                              限制= 0 = False
    MultiSpriteCounter类型=门户类型2=alien limit=0 win=True

  LevelMapping
     .  >  背景
    0 > background base
    1 >后台portalSlow
    2 > background portalFast
    A>背景头像
```

清单1:VGDL游戏外星人的定义，灵感来自经典游戏太空入侵者

在第一行，关键字BasicGame标志着标准VGDL游戏定义的开始(第7章将研究其他类型的游戏)。在本例中，它指定了⊠es一个参数，即正方形大小，它决定了游戏棋盘中每个单元格的像素大小。这纯粹是一个美学方面的问题。

第2行到第14行是精灵集。《异形》需要六种不同类型的精灵(背景，基地，化身，导弹，外星人和传送门)，其中一些还有子类型。每个精灵对应的类是每个类型或子类型上以大写字母开头的关键字。例如，化身精灵是of

类型MovingAvatar，而两种类型的门户(portalSlow和portalFast)将是SpawnPoint的实例。可以看到，tpye的de⊠nition既可以在一个层级的父级(如异形，第9行)，也可以在子级(如传送门，第13行和第14行)。

这些精灵接收到的一些参数对所有精灵来说都是共同的。《异形》中的例子是img，它描述了一个将在游戏中以图像形式呈现精灵的角色，或者Singleton，它将向引擎表明该精灵在任何时候只能出现一个实例。在这款游戏中，这个精灵就是sam(玩家射击的子弹):在原版游戏《太空入侵者》中，玩家一次只能射击一颗子弹。另一个有趣的参数被隐藏了，它向引擎表明，这个精灵一定不能包含在智能体接收到的观察中(参见第3节)。这在GVGAI框架中用于整个游戏中建模游戏中的部分可观察性。一个di⊠erent参数，不可见，只隐藏了游戏的视觉表现中的精灵，但在游戏观察中仍然是可访问的。

然而，许多参数取决于精灵的类型。例如，异形的角色被设置为FlakAvatar，本体将其定义为一种角色类型(即由玩家控制)，只能向左和RIGHT移动，并在执行动作USE时生成一个对象，该对象的类型在参数类型中需要。在这种情况下，stype以精灵sam作为值，也就是下面代码中的de⊠ned。

Sam以导弹为母精灵。导弹属于导弹类型，是一种在⊠nitum中以直线方向ad移动的精灵类型。在《异形》中，有两种类型的导弹:萨姆(玩家射击)和炸弹(外星人投掷)，它们具有不同的导弹类型参数。其中一个参数是方向(orientation)，它决定了精灵的行进方向。另一个参数speed表示精灵每帧移动多少个单元格。从第7行和第8行可以看出，这些飞弹具有di⊠不同的方向和速度(如果参数speed不是de⊠ned，则采用默认值1.0)。

玩家必须消灭的敌人在第9行到第11行进行了描述。外星人是一个等级体系的父精灵，⊠将他们视为轰炸机类型。轰炸机是朝着特定方向移动的精灵，同时生成其他精灵。在导弹的情况下，它们的移动是由速度和方向(默认设置为RIGHT)决定的。此外，参数cooldown描述了精灵执行一个动作的速率(在例子中，外星人每3帧移动或射击一次)。参数stype描述了以概率(每个动作)prob生成的精灵类型，如第9行中的de⊠ned。

异形是由传送门产生的(第13到14行)，传送门可以被看作是不可移动的轰炸机。他们可以de⊠ne最大的精灵产卵数量，之后

刷出者被摧毁，使用参数total。在这种情况下，概率prob默认设置为1.0，所以外星人的生成速率只由冷却参数来设置。

精灵集可以说是VGDL中最复杂和最丰富的集合，但如果没有描述它们之间互动的规则，就不能创造出一款游戏。第16 ~ 26行描述了这一点的交互设定。交互集中的每一行描述了两个精灵相互碰撞时发生的e⊠ect。为了保持游戏的描述简短，可以用两个或更多的精灵在指令中分组成对的交互(见第25行)。一个类型的指令$A_1A_2A_3..A_N$> E相当于N E⊠ectsA_1A_2> E，A_1A_3>E..A_1A_N> E等。

前三个交互(第17行到第19行)描述了当精灵离开游戏屏幕时应用的效果(关键字EOS -屏幕结束)。当角色到达屏幕尽头时，游戏会自动执行事件stepBack，将精灵的位置设置为之前的位置(因此阻止了它离开屏幕)。然而，当外星人到达终点时，执行的事件是周转(turnAround)，它将精灵向下移动一个单元格，⊠ips其方向。这使得敌人在关卡中下降，同时保持他们的横向移动。最后，当导弹到达EOS时，它们被摧毁。注意，这种交互作用会影响精灵集中需要的两种导弹(sam和bomb)，并且对于角色能够再次射击至关重要(因为sam精灵是单一的)。

这个游戏中的其他互动会调节什么时候精灵应该被摧毁。当与炸弹、萨姆和外星人碰撞时，基地(结构防御)会被摧毁。在⊠rst两种情况下，另一个精灵也被摧毁(通过e⊠ect击杀两者)。化身在与外星人或炸弹(敌人的导弹)碰撞时被摧毁。最后，外星人被sam(玩家的导弹)击中时被杀死。

交互设定的一个重要职责是de⊠ne分数系统，而分数系统又可以作为学习算法的奖励。每次交互都可以de⊠ne一个参数scoreChange，它建立了每次触发交互时奖励多少分。因此，在《外星人》中，当玩家摧毁基地(第22行)时将获得+1分，当摧毁外星人(第26行)时将获得+2分，当角色被摧毁时将失去1分(第25行)。

终止集(第28行至第30行)描述了游戏可以以何种方式结束。所有的指令都必须de⊠ne这个参数win，它从玩家的角度决定了游戏的胜负。⊠rst条件(第29行)的类型是SpriteCounter，它检查一种特定类型的精灵的数量是否达到极限。在这种情况下，这条指令决定了，当玩家被杀死时，游戏以失败告终。第二个条件(第30行)，MultiSpriteCounter,

要求两种类型的精灵(stype1和stype2)在游戏中都没有实例，游戏才会结束。在这种情况下，当游戏中没有外星人，传送门也消失了(即它已经产生了所有外星人)，游戏⊠以玩家的胜利而结束。

最后，关卡映射集描述了关卡⊠le中对应角色de⊠ned的精灵。所有的VGDL游戏都可以在许多关卡中进行，这些关卡在游戏开始时设置了精灵的初始配置。图1显示了框架中可用的两个外星人关卡示例。

```
1
000.....
000.....
.....
.....
.....
.....
....000.....000000.....000...
...00000....00000000...00000..
...0...0....00....00...00000...
```

```
2
000.....
000.....
.....
.....
.....0000.....0000.....
.....0..0.....0..0.....
.....
.....
..00000....00000000....00000..
```

图1:《外星人》的两个VGDL关卡

关卡de⊠nition和mapping中的每个角色都可以de⊠ne一个或多个将在该位置创建的精灵。在《外星人》中，所有角色都有一个共同的精灵:背景。这个精灵，在精灵集中是不需要的，它的唯一目的是在所有精灵后面绘制(所以，例如，当一个基地被破坏时，就会绘制一些东西)。还要注意的是，拥有两个di⊠erent类型的传送门，可以用di⊠erent di⊠culties创造di⊠erent关卡。portalFast(角色⊠2⊠)比portalSlow (respp .)生成外星人的速率更高(使游戏更di⊠cult)。⊠⊠1)。图1中的两个关卡使用di⊠erent门户(左边的关卡比较简单的情况，右边的关卡比较困难)。图2显示了《外星人》游戏在中间阶段的截图。

VGDL是一种丰富的语言，允许定义多个二维游戏，其中精灵可以主动移动(通过本体中指定的行为或玩家的动作)，被动移动(受物理影响)，并与触发其效果的其他物体碰撞。这些交互是两个碰撞物体之间的局部交互，这可能导致一些精灵消失、生成、转化为di⊠erent

图2:《GVGAI》游戏《外星人》截图

类型或改变属性。非局部交互也可以实现，如传送事件或e⌧ects，⌧等多个精灵(所有，或按类型)是可能的。

基于本体的VGDL允许描述简洁。这个本体de⌧nes游戏的高级构建模块，如物理、运动动态和交互。有了这些工具，VGDL就能够描述各种各样的街机游戏，包括《太空入侵者》、《Sokoban》、《Seaquest》、《Pong》、《吃豆人》、《Lunar Lander》或《Arkanoid》等经典街机游戏的版本。

然而，其中一些游戏不能在原始VGDL中用于GVGAI。下面两个章节描述了对现实世界物理的介绍(2.1节)，包括e⌧等引力和惯性，以及de⌧ne 2人游戏的能力(2.2节)。

2.1 连续物理

最初的VGDL可以为GVGAI描述的游戏类型仅限于具有离散物理的街机游戏。2017年之前的所有游戏都是基于离散化游戏状态和可用动作的网格结构。例如，上面描述的精灵的方向可以是Nil，上，下，左或右，而不提供⌧ner精度。虽然这对于创造像《异形》这样的游戏来说已经足够了，但是对于像《Lunar Lander》这样的游戏来说就不够了，因为后者的定位和移动都需要⌧ner的精度。

Perez-Liebana等人[5]在2017年引入了VGDL的更新，该更新增强了本体，以扩大可以构建的游戏范围。对框架的主要补充是建立一个连续的物理设置的能力，这将允许精灵以更⌧ne-tuned的方式移动。精灵现在可以被要求

通过为CONT设置一个新的参数物理类型来坚持新的物理类型。与这些物理类型对齐的精灵获得三个额外的参数，这些参数可以被需要:质量，摩擦和重力。虽然将重力设置为与所有精灵相等是正常的，但也可以为它们单独设置。

精灵的移动仍然是活跃的，就像在传统网格物理的情况下一样，但连续物理不会将移动限制在四个方向上。相反，方向可以设置为360◦。此外，速度不是de⊠ned成比例的细胞，而是屏幕像素，允许精灵移动在任何可能的方向和速度。

精灵也会受到被动移动的影响，这在每个游戏帧中都会发生。这种被动移动会改变精灵的方向，可能是摩擦力，也可能是重力。如果精灵是一个⊠被摩擦作用的物体，它的速度将乘以1−f(s)(其中f(s)是精灵的摩擦力)。如果它响应重力，则一个指向下方的矢量-→g，其大小等于精灵的质量乘以精灵的重力，将其添加到精灵当前的速度矢量中。

图3显示了一个使用连续物理(Arkanoid)构建的游戏示例。在这个游戏中，玩家控制着屏幕底部的球棒。它必须弹起一个球，摧毁关卡中的所有砖块，才能获胜。当球掉入底部时，玩家就失去一条命，当所有的命都输掉时，玩家就输掉游戏。

清单2显示了这个游戏的精灵设置的simpli⊠ed版本。化身精灵，类型为FlakAvatar(第5行，de⊠nes物理类型是连续的。摩擦和惯性被设置为0.2，这建立了蝙蝠运动的动力学。其他有趣的参数是de⊠ne玩家的生命数(healthPoints和limitthealthpoints)和蝙蝠宽度(wMult)。

球(导弹型)也遵循这种物理类型。对于这个游戏，不需要重力(Arkanoid中的球并不真正遵守物理定律)，但速度需要达到每秒20次。

图3:游戏Arkanoid在GVGAI中的截图。

```
SpriteSet
        background > Immovable img=oryx/spacel hidden=True

    FlakAvatar类型=球物理类型=CONT wMult=4摩擦=0.2
        质量=0.2 img=oryx/floor3 healthPoints=3 limitthealthpoints =3

    球>导弹方向=向上速度=20物理类型=CONT
    ballLost >被动隐形=真隐藏=真

    brick >被动img=newset/blockG wMult=2
    block >被动img=newset/block2 wMult=2
        清单2:游戏Arkanoid的VGDL Sprite Set block。l
```

2.2 双人游戏

最初的VGDL只允许创建单人游戏。al.[2]和[1]的Gaina引入了de⊠ning 2人同时游戏的可能性，使用这种语言。

语言的□rst modi□阳离子之一是增加了一个新参数(no players)来指定游戏的玩家数量。这被添加到游戏描述的□rst行中，如清单3所示。

```
BasicGame square_size=50 no_players=2
```

Listing 3: Specifying the number of players in VGDL.

此外，分数交互必须指定每个玩家在触发事件时收到的点数，终止指令必须确定每个玩家的结束状态(赢或输)。在这两种情况下，积分之间用逗号分隔，如清单4所示。

```
avatarA npc > killSprite scoreChange=-1,0
avatarB npc > killSprite scoreChange=0,-1

SpriteCounter stype=avatarA limit=0 win=False,True
SpriteCounter stype=avatarB limit=0 win=True,False
```

Listing 4: Score and termination instructions for a 2-player VGDL game.

需要注意的一个重要方面是，游戏可以是合作的，也可以是竞争的，这是从描述游戏的交互和终止集推断出来的。图4显示了GVGAI中2人游戏的两个例子。左边的是《扫雷者》，在这款游戏中，两名玩家将在关卡中分散的箱子中寻找自己的炸弹。在右侧的推箱子中，两名玩家合作将所有箱子推入目标洞中。

2.3 VGDL的未来

截至2019年7月，已推出140款单人游戏和60款双人VGDL游戏，每款游戏有5个不同的关卡。在本书的其余部分可以看到，VGDL已经成为GVGAI的关键部分，作为一种快速原型和设计新挑战的语言。连续物理和双人玩家能力的融合，拓宽了可以创造的可能游戏的种类。

然而，VGDL仍然是有限的。尽管VGDL具有表现力，但我们却很难创造出真正让人类感到有趣的游戏。此外，创造复杂的游戏在语言上也受到限制，这反过来又限制了智能代理可以提出的挑战类型。除了GVGAI与非VGDL游戏的接口(见第8章)之外，还设想了VGDL的进一步工作。

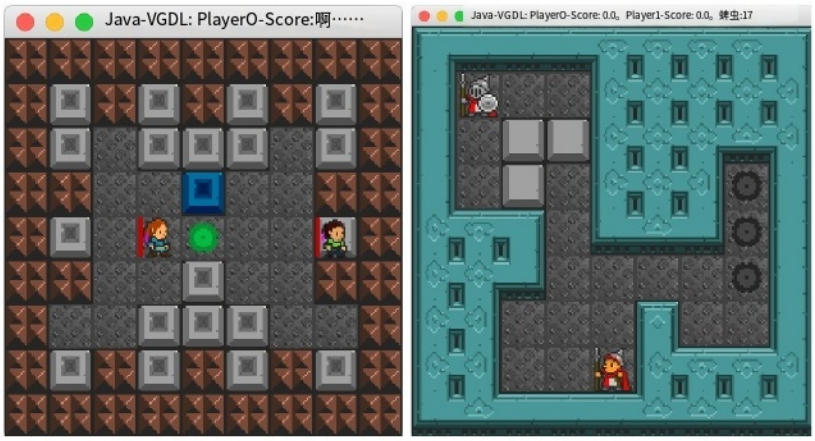

图4:两款双人VGDL游戏示例:《扫雷》(左)和《推箱子》(右)

VGDL的一个潜在变化是将其用于玩家不一定需要控制角色的游戏(这是当前版本的要求)。例如，像《俄罗斯方块》或《糖果粉碎传奇》这样的游戏，如果没有大量的指令和黑客操作，就无法在当前版本的VGDL中创造出来。另一个合乎逻辑的下一步是走向创造3D游戏的可能性，以及从2人模式转向n人挑战模式。

我们将在第6章和第7章中看到，VGDL用于内容生成和游戏设计。VGDL最初并不是为这类任务设计的，因此对其进行调整并不是一件容易的事。向更模块化的语言发展可以简化规则、关卡和游戏的自动设计，并促进新的有趣的游戏和挑战的创造。

3 通用的视频游戏AI框架

VGDL的原始实现是在py-vgdl中解析的，py-vgdl是Schaul编写的一个Python解释器[7]，它允许代理玩原始的VGDL游戏。为了给规划代理提供一个快速前进的模型，py-vgdl被移植到Java中作为第一个GVGAI框架和竞争。这个框架能够加载VGDL中不需要的游戏和关卡，并为机器人提供访问前向模型的API。图5显示了工作中的GVGAI框架的方案。

Java-vgdl引擎解析游戏和关卡描述来创建两个组件:一个游戏对象和一个控制器，它可以是一个由机器人或a

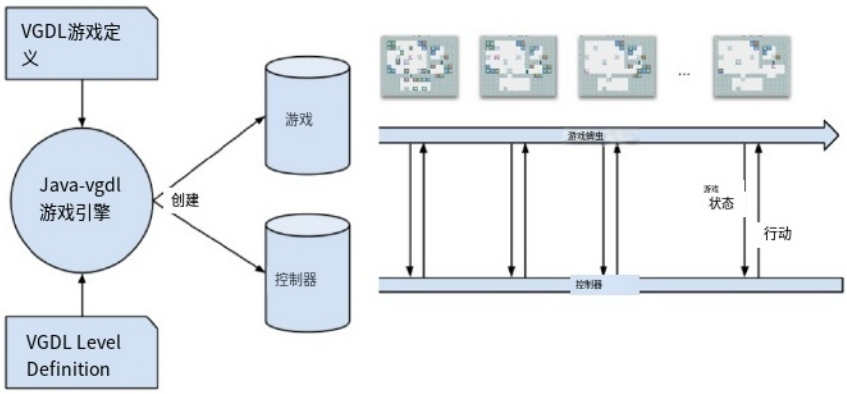

图5:GVGAI框架

人类的球员。然后，游戏开始，并在每个游戏帧从控制器请求一个动作。控制器必须在不超过40ms的时间内返回一个在真实游戏中执行的动作，否则将适用处罚。如果操作在40到50毫秒之间返回，则将应用空操作(NIL)。如果动作在超过50ms后获得，则控制器被取消资格□ed。设置这个规则是为了保持游戏的实时性。

GVGAI机器人必须根据给定的API实现类。在单人游戏中，该类必须继承抽象类AbstractPlayer(双人游戏的abstract多人模式)。单玩家游戏控制器的API包含在清单5中。

```
1  public <ClassName>(StateObservation, ElapsedCpuTimer);
2
3  Types.ACTIONS act(StateObservation, ElapsedCpuTimer);
4
5  void result(StateObservation, ElapsedCpuTimer);
```

Listing 5: API for single-player games.

函数act在游戏的每一帧被调用，并且必须返回一个类型。每40毫秒返回一个ACTIONS值。虽然每个游戏都有不同的可用动作集，但所有游戏都选择所有可能移动的子集:右、左、上、下、USE和NIL。前四个是移动动作，USE被用作非移动动作的通用卡，角色类型可能会取消这些动作(例如，射击)，NIL用于返回一个无效动作。第二个函数result是可选的，只有在游戏结束时才会调用。

然而，这个函数act必须在bot中实现。这两个函数都接收两个参数:一个状态观察和一个定时器。定时器可以通过查询来控制剩下的时间，直到需要返回动作。状态观察提供了关于状态的信息:

-当前的时间步长、分数、胜利状态(胜利、失败或正在进行)，以及玩家的生命值。
-玩家可用动作列表。
-观察列表游戏中的每个精灵都被分配一个唯一的整数类型和一个与其行为相关的类别(静态、非静态、非玩家角色、收藏品和门户)。对精灵的观察，除非是隐藏的精灵，包含关于它的类别、类型id和位置的信息。这些观察结果被提供在一个列表中，按类别(⊠rst)和精灵类型(second)分组。
-一个以网格格式包含所有可见观测的观测网格。在这里，每个观测都位于一个二维数组中，对应于关卡中精灵的位置。
-化身事件的历史。化身事件是化身(或由化身产生的精灵)与游戏中任何其他精灵之间的互动。化身事件的历史被提供在一个排序列表中，按照游戏步骤进行排序，每个实例提供了关于交互发生的地点、每个事件的类型和类别的信息。

状态观察提供的最重要的功能之一是前向模型(FM)。来自状态观察的advance函数接收一个动作，并应用该动作将游戏状态向前推进一步。下一个状态将是智能体将⊠nd的可能的下一个状态之一，如果该动作被应用于真实的游戏中。游戏本质上是随机的，所以除非游戏是完全确定的，否则下一个状态将从可能存在的可能的下一个状态中采样。另外，状态观察提供了copy函数，它允许对当前状态进行精确的复制。这两个函数对于能够运行蒙特卡洛树搜索(MCTS)或滚动地平线进化算法(RHEA)等统计前向规划(SFP)算法是必不可少的。

除了函数act之外，控制器还必须实现一个接收相同参数的构造函数。这个构造函数构建代理，并在游戏开始前被调用。这个函数必须⊠nish在1s，可以用来分析游戏和初始化代理。值得注意的是，智能体永远无法访问游戏的VGDL描述——它必须猜测规则、动态和使用FM获胜的方法。

3.1 双人游戏的API

为双人游戏创建的bot需要遵循不同的API。这些是主要的di erence。

-实现代理的类必须继承基类abstract多人模式。
-在单人游戏的情况下，同样的功能是可用的，尽管在这种情况下，状态观察是作为一个StateObservationMulti对象接收的。
-每个玩家接收一个整数形式的玩家ID，它作为构造函数中的第三个参数提供。这个玩家ID可以是0或者1，它是由游戏引擎分配的。清单6显示了2人游戏代理的API。

```
1    public <ClassName>(StateObservationMulti, ElapsedCpuTimer, in t);Types . ACTIONS act
2    (StateObservationMulti, ElapsedCpuTimer);(StateObservationMulti, ElapsedCpuTimer);
3
4                                清单6:单人游戏的API。
5
```

分数，胜利状态，生命值和可用动作列表仍然可以通过StateObservationMulti对象获得。但是，现在各自的方法会收到一个参数:来自被请求信息的玩家的玩家ID。请注意，代理可能有一组di erent可用的动作。

前进模型在单人模式下是可用的，但是前进功能现在接收游戏中所有玩家的一系列动作。这个数组中的索引对应于玩家ID，动作同时执行以向前滚动状态。

在2人游戏中，代理可能有di 不同的目标和赢得游戏的方法。API并不提供关于游戏的合作或竞争性质的信息，而是将这些信息留给玩游戏的代理去发现。

3.2 GVGAI是一个多轨道挑战

GVGAI框架的最初用途是玩单人游戏计划，后来扩展到双人游戏。我们将这种设置(或轨迹)称为规划，因为前进模型(FM)的存在允许实现统计前向规划方法，如蒙特卡洛树搜索(MCTS)和滚动地平线进化算法(RHEA)。智能体不需要之前的游戏训练，但FM允许机器人分析搜索树

以及具有正确游戏模型的动作序列。第3章展示了可以用于GVGAI规划挑战的简单和高级方法。

另一个不同的任务是学会在没有FM的情况下间歇性地玩游戏。学习轨道后来被添加到框架中，以解决这种类型的学习问题。对于这项任务，除了Java之外，agent也可以用Python编写。这是为了适应用Python开发的流行机器学习(ML)库。除了FM之外，关于游戏状态的相同信息也会提供给代理，但在这种情况下，它是以JavaScript Object Notation (JSON)格式提供的。此外，还可以观察游戏屏幕，仅通过像素进行学习任务。Torrado等人[8]将GVGAI框架连接到OpenAI Gym环境，以便在ML社区中进一步扩展。第5章更深入地探讨了GVGAI的这种设置。

GVGAI框架主要用于玩游戏的代理，但游戏本身也可以作为目的。特别值得一提的是，该框架被增强了用于自动关卡和规则生成的接口。这些是程序内容生成(Procedural Content Generation, PCG)轨道，其目标是创建生成器，为收到的任何游戏生成关卡，并(分别)创建规则，根据给定的任何关卡和精灵集形成可玩游戏。在这两种情况下，都授予对FM的访问权限，因此生成器可以使用规划代理来评估生成的内容。第6章更详细地描述了这些接口。

因此，GVGAI对游戏AI的多个领域提出了挑战，每个领域都包含上述的一种设置。规划跟踪了使用基于模型的方法(如MCTS和RHEA)的可能性，并在2人游戏的特殊情况下，提出了对一般对手建模和与另一个代理交互等问题的研究。学习轨道促进了对通用无模型技术和类似方法的研究，如神经进化。最后但并非最不重要的是，内容生成设置将重点放在PCG任务通常使用的算法上，例如求解器(满足能力问题和答案集编程)，基于搜索的方法(进化算法和规划)，基于语法的方法，细胞自动机，噪声和分形。

4 GVGAI竞赛

前面章节中描述的每一种设置都是游戏AI社区的竞争轨道。表1总结了截至2018年的所有版本的比赛。

单人规划

比赛	赢家	方法	% 胜利	条目
2014年的是到岸价	阿德里恩克特克斯	开环期望值树搜索(OLETS)	51.2%	14
2015年GECCO	YOLOBOT	MCTS, BFS，精灵目标启发式	63.8%	47
CIG 2015	Return42	遗传算法，随机漫步，A*	35.8%	48
CEEC 2015	YBCriber	迭代加宽，危险规避	39.2%	50
GECCO 2016	接地 CTS2	增强其特定	43.6%	19
CIG 2016	YOLOBOT	(见上图)	41.6%	24
2017年GECCO	YOLOBOT	(见上图)	42.4%	25
WCCI 2018	asd592	遗传算法，BFS	39.6%	10

两人计划

比赛	赢家	方法	点	条目
WCCI 2016	ToVo2	带td备份的SARSA-UCT	178	8
CIG 2016	阿德里恩克特克斯	OLETS	142	8
CEC 2017	ToVo2	(见上图)	161	12
FDG 2018	adrienctx	(见上图)	169	14

水平一代

比赛	赢家	方法	%的选票	条目
IJCAI 2016 国际商事展览会	easablade	元胞自动机	36%	4
CIG 2017	(暂停)	-	-	1
2018年GECCO	架构师	建设性和进化生成器	64.74%	6

规则生成

比赛	冠军(暂停)(	方法	%的选票	条目
CIG 2017	暂停)	-	-	0
2018年GECCO		-	-	0

表1:GVGAI竞赛的所有版本，包括获奖作品、参赛方式、成绩和提交作品数量(样本代理除外)。所有版本在一套di☒erent的游戏中运行，一些竞赛获胜者没有参加所有版本。

除非特别说明☒ed di☒erently，所有的赛道都遵循类似的结构:游戏被组织成10个集合(每个游戏加5个关卡)。N个集合[1]是公开的，可以在框架代码[2]内使用。因此，参与者可以使用这些游戏来训练他们的代理。

一个额外的集合用于验证。它的游戏是未知的，不提供给参赛者，尽管他们可以将他们的代理提交到竞赛服务器[3]，在他们中进行评估(以及在训练集中)。服务器提供排名

[1] N depends on the competition. The collection of GVGAI games grows by 10 (1 set) for each successive edition.

[2] Competition Framework: https://github.com/GAIGResearch/GVGAI

[3] Competition website: http://www.gvgai.net/

基于训练和验证集的代理，保持验证游戏匿名。最后，使用隐藏和秘密的测试集对代理进行评估，并形成竞赛的⌧nal排名。在提交关闭之前，参赛者不能访问这些游戏，也不能对他们的代理进行评估。一旦比赛结束，验证集的游戏和级别将被发布并添加到框架中，测试集将被用作下一版比赛的验证，并将创建10个新游戏以形成下一个测试集。

比赛排名，在一组比赛中，计算如下:首先，所有参赛选手在所有可用级别中玩每个游戏(一次用于训练集和验证集，5次在测试集)。如果在此之前没有触发自然游戏结束条件，那么所有比赛在2000帧之后结束，玩家会输一场。游戏排名是通过将所有参赛作品⌧rst按胜利的平均值排序，使用平均分数和游戏时长作为平局决胜局，按此顺序进行排序。除了后者，所有的指标都是要最大化的。对于时间步数，t表示以胜利告终的游戏，2000 - t表示以失败告终的游戏;这奖励游戏赢得快，输得晚。

对于每一场比赛排名，按照F1系统给代理奖励积分:从⌧rst到第十名，分别给予25、18、15、12、10、8、6、4、2和1分。从第10位开始，不给积分。⌧nal的排名是通过将每局比赛中取得的分数相加来计算的。通过计算游戏排名中⌧rst位置的数量来打破最终的排名平局。如果平局持续，则考虑取得的第二名的数量，以此类推，直到平局被打破。

在2人计划的情况下，⌧nal的排名是通过在⌧nal集合中的所有提交的代理中进行轮转锦标赛来计算的，在时间允许的情况下进行尽可能多的迭代。在这种情况下，所有关卡都要打两次，交换代理的位置，以考虑到游戏中可能出现的不平衡。训练和验证集运行di⌧显然，尽管如此，如果每次收到新提交时都需要运行整个循环，则不可能提供快速反馈。在这种情况下，我们建立了一个Glicko-2评分系统[3]，用于选择接下来必须上场的两个agent。在这些集合中，Glicko-2分数成为形成排名的第一个指标。测试集也提供了Glicko-2分数，但只是作为一个额外的指标:在计划情况下计算最终排名(胜利，分数和游戏持续时间)。

学习轨道已经运行了两个di⌧erent版本(2017年和2018年)。对于2017年的案例，控制器的运行分为两个阶段:学习和验证。在学习阶段，每个条目都有有限的时间(5分钟)用于培训

□rst每场比赛三个等级。在验证步骤中，控制器玩的游戏是每局其他两关的10倍。

2018年比赛中使用的框架[4]与OpenAI Gym接口，比赛运行时有一些放松的约束。比赛中使用的套装只有三场比赛，并且都是公开的。其中两个等级是提供给参赛者训练的，另外三个则是保密的，用于获得□nal的成绩。此外，每个智能体可以依靠100ms作为决策时间(而不是传统的40ms)。关于学习轨迹设置的完整细节，读者可以参考第5章。

最后，PCG赛道的优胜者是由对参赛作品生成的内容进行评估的人类受试者决定的。在这两种设置中，评委都会得到成对的生成内容(关卡或游戏)，并被问及哪一个(或两者都有，或都没有)最受欢迎。获胜者是根据投票更多的生成器选出的。更多关于内容生成轨道的细节，请阅读第6章。

参考文献

1. R. D. Gaina, A. Cou¨etoux, D. J. Soemers, M. H. Winands, T. Vodopivec, F. Kirchgessner, J. Liu, S. M. Lucas, and D. Perez-Liebana, □The 2016 Two-Player GVGAI Competition,□ IEEE Transactions on Computational Intelligence and AI in Games, 2017.
2. R. D. Gaina, D. Perez-Liebana, and S. M. Lucas, □General Video Game for 2 Players: Framework and Competition,□ in IEEE Computer Science and Electronic Engineering Conference, 2016.
3. M. E. Glickman, □Example of the Glicko-2 system,□ Boston University, 2012.
4. N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi, □Illuminating Gener-alization in Deep Reinforcement Learning through Procedural Level Generation,□ arXiv:1806.10729, 2018.
5. D. P´erez-Li´ebana, M. Stephenson, R. D. Gaina, J. Renz, and S. M. Lucas, □Introducing Real World Physics and Macro-Actions to General Video Game AI,□ in Conference on Computational Intelligence and Games (CIG). IEEE, 2017.
6. T. Schaul, □A Video Game Description Language for Model-based or Interactive Learning,□ in IEEE Conference on Computational Intelligence in Games (CIG), 2013, pp. 1–8.
7. ——, □A Video Game Description Language for Model-based or Interactive Learning,□ in Proceedings of the IEEE Conference on Computational Intelligence in Games. Niagara Falls: IEEE Press, 2013, pp. 193–200.
8. R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Perez-Liebana, □Deep Reinforcement Learning in the General Video Game AI framework,□ in IEEE Conference on Computational Intelligence and Games (CIG), 2018.

[4] GVGAI-Gym Learning track framework: https://github.com/rubenrtorrado/GVGAI_GYM