第8章-不带VGDL的GVGAI

西蒙·m·卢卡斯

1 介绍

在本章中，我们概述了用高级语言而不是使用VGDL编写GV-GAI兼容游戏的动机和主要原则。下面描述的特定示例都是用Java编写的，但是这些原则同样适用于其他高级语言。Java，包括它的图形api，可以在Windows, macOS和Linux上运行，而不需要任何额外的库或插件。我们目前正在评估使用新的Kotlin[1]编程语言作为Java更好的替代品。Kotlin具有许多语言特性，可以使代码更清晰、更简洁，同时与Java平台完全可互操作(Kotlin代码编译为Java虚拟机- JVM -代码)。Kotlin还有其他优点，比如对编写领域特定语言的强大支持[3](这将支持用于描述特定游戏变体的灵活语言，或开发新版本的VGDL)。此外，如果所有的代码都是纯Kotlin，并且不调用任何Java或本机库，那么Kotlin也可以被翻译成JavaScript，然后在web浏览器中运行。虽然过去在web浏览器中运行Java小程序是可能的，但近年来这已经变得越来越不受欢迎，大多数web浏览器安全管理器都在阻止它。

首先是动机。对于表达相对简单的游戏来说，VGDL是一个不错的选择，在这种游戏中，游戏对象的相互作用及其触发的效果可以通过包含基本计算的条件操作规则来表达。当这些条件保持不变时，游戏可以用易于理解的简洁代码快速编写，并描述游戏的核心元素，同时避免任何干扰。

然而，VGDL的限制性使得执行具有复杂和战略深度玩法的游戏变得相对困难。此外，现有的VGDL游戏引擎的运行速度要比用Java、c++或c#等高级语言精心实现的游戏慢得多。使用高级语言而不是VGDL可能会使游戏速度提高100倍。速度的提高很重要，不仅可以更快地运行实验，还可以使统计前向规划(SFP)方法成为可能

[1] https://kotlinlang.org/

在更大的模拟预算下实时运行。这使得智能体以更智能、更有趣的方式表现，并提供对游戏技能深度的更好估计。

另一个因素是程序员的偏好:尽管VGDL很优雅，但许多程序员仍然喜欢用更传统的高级语言编写游戏。

GVGAI游戏的设计主要是为了测试AI代理播放它们的能力，在大多数GVGAI曲目的情况下，并在关卡生成曲目中为它们设计内容。在判断AI玩法和生成内容的质量时，让它们具有人类可玩性也很重要。这让我们能够比较AI与人类玩法的强度，也让玩家能够评估生成的游戏和内容。尽管最近的努力以许多有趣的方式扩展了VGDL，例如添加连续物理，但该语言在游戏规则和交互模式方面仍然存在一些需要克服的障碍。例如，在默认情况下，飞船角色可以在任何特定时间移动或射击，但人类玩家通常希望两者都做，目前GVGAI不支持任何点击输入设备，如鼠标或触控板。

基于vgdl的游戏是围绕一定数量的关卡设计的，以测试智能体的泛化能力。一个更好的选择是允许随机或其他程序化的关卡生成方法，这在高级语言中更容易实现。

请注意，GVGAI的目标是真正通用的游戏AI，可以在多个维度上进行彻底的测试，并且在挑战的可变性和挑战的深度方面，有可能远远超过雅达利学习环境中使用的雅达利2600游戏[1]。

因此，在GVGAI中使用高级语言的好处是显而易见的。对于游戏设计师或AI研究人员来说，制作一款与GVGAI兼容的游戏的优势也是非常显著的:只需要一点点额外的努力，它就可以让许多代理进行测试或玩游戏。经过[6]的努力，GVGAI游戏现在可以在OpenAI Gym中使用[2]，这对于本章中的示例游戏以及符合相同接口的未来游戏来说也是可能的。

最后，我们实现的每一款游戏真的应该被认为是一类游戏，有很多可变参数来控制游戏玩法。这有助于证明实现Java游戏所涉及的额外工作是合理的，因为它可能涉及广泛的测试用例，不同的参数设置导致非常不同的游戏玩法。

2 实施原则

在本节中，我们将概述实现这些游戏时需要考虑的主要因素。下面所采取的步骤并不是di⌧邪教，并且有助于引导出设计良好的软件。当从头开始执行一款游戏时，这种方法是最容易采取的——复古⌧将这些设计原则应用到现有的实现中是非常困难的。

2.1 复制游戏状态

在所有的GVGAI赛道中，规划赛道一直是最受欢迎的，尤其是单机版的规划赛道。规划航迹兼容性的主要要求是能够复制当前的游戏状态，并且推进速度要比实时快得多。在这些要求中，最需要实现的就是可复制性。对于一个结构良好的面向对象设计，游戏状态通常会包含一个由许多类的许多对象组成的对象图。这并不能表现出任何伟大的di⌧culty，但它确实意味着必须小心翼翼地对游戏状态进行深度复制。此外，必须避免使用静态或全局变量。循环引用也是最好避免的，因为它们会导致将游戏状态序列化为JavaScript Object Notation (JSON)的问题，因为JSON不支持循环引用。

复制游戏状态有两种方法:可以使用通用的序列化器/反序列化器，如GSON[2]或WOX[3]，但请注意，GSON要求游戏状态对象图不得包含任何循环引用(这是JSON的限制，而不是GSON库本身)。注意，序列化游戏状态也很有用，这样可以保存特定的状态以供将来参考，也可以允许代理通过网络连接与游戏进行交互。通用选项允许复制功能在几行代码中实现，但它们不是很⌧灵活。它们本质上比定制的复制方法慢，因为它们使用比常规方法调用慢的运行时re⌧ection操作符。

因此，编写定制的游戏状态复制方法通常是可取的。除了提供更快的速度之外，它们还具有⌧灵活性，如果需要，可以只对大型不可变对象进行浅复制。例如，一个游戏可能包含一个大的地理地图，在游戏过程中不会发生变化。在这种情况下，制作一个浅复制(即只复制地图的对象引用)是su⌧cient和更快的，也节省了内存分配/垃圾回收[4]。

[2] https://github.com/google/gson

[3] http://woxserializer.sourceforge.net/

[4] The potential value of writing bespoke copy methods is clear when using Java, but may be unnecessary when using Kotlin. Kotlin has special data classes which provide default copy methods, and it also has immutable types.

当执行一款游戏时，通常会有许多参数与游戏玩法相关。参数设置的特定组合可以在潜在的广阔游戏空间中创造出一款游戏。这个game-space视图有两个bene⊠ts。我们可以通过从空间中选择多个游戏来更彻底地测试代理，以确保它们没有结束-⊠tting到游戏的特定实例。此外，我们可以使用优化算法来调整游戏以满足特定的目标。为了最大化⊠灵活性，我们建议将每个游戏参数存储在单个对象中，de⊠ned在一个数据绑定类中。例如，假设我们有一个PlanetWars游戏，我们然后de⊠ne一个名为PlanetWarsParams的类。这个类的每个对象de⊠ne都是游戏参数的一个特定组合，因此是这个空间中可能的游戏集合中的一个特定游戏。

在复制游戏状态时，还必须对参数对象进行深度复制。这样就可以同时实验di⊠不同的游戏版本。一个有趣的用例如下:SFP代理可以被赋予一个假FM，即其中的参数与测试游戏中使用的参数相比发生了变化。使用假前向模型进行实验，可以提供有价值的见解，以了解一个模型在变得无用之前可以有多糟糕。有趣的是，初步测试表明，模型可以在保持有用的同时高度不准确。

2.2 推进游戏状态

SFP算法对下一个状态函数的调用通常比对复制方法的调用要多。例如，具有L序列长度(滚出深度)的滚动地平线演化或蒙特卡洛树搜索将使L因子比游戏状态复制更多的下一个状态调用。对于电子游戏来说，L的值通常在5到500之间变化，因此这意味着实现e⊠cient next state函数时必须谨慎。

除了遵循正常的良好编程实践，主要的实现方式如下。

-使用⊠cient数据结构来检测物体碰撞。。
-预先计算和缓存将定期需要的数据。
-在对象更新循环中，在每个游戏周期更新每个对象可能是不必要的，甚至是不希望的。

用于检测对象碰撞的最简单和最简单的数据结构是一个数组，它可以被安排为2d游戏的2d网格和o⊠ers常量时间访问(访问一个网格元素的时间不取决于网格的大小或其中对象的数量)。对于3d游戏来说，二叉空间分割树是一种常见的结构

选择，o⊠相对于条目数量具有对数成本的访问。

一个预计算的例子可以在我们实现[4]的PlanetWars的扩展版本中找到。每个行星都会对过境中的飞船施加引力，但这被预先计算为一个向量⊠eld并存储在一个2d数组中，当有许多过境的飞船时，将下一个状态函数的速度提高到行星数量N的一倍。

关于对象更新，有些游戏甚至会使用跳过更新来提升游戏玩法。例如，Taito(1979)的《太空入侵者》(Space Invaders)在每个游戏tick中只移动一个外星人。这导致了两个突发的、有趣的e⊠ects。一种是外星人以一种视觉上吸引人的狗腿方式移动，只是偶尔排成一排。另一种是，当同胞被消灭时，剩下的外星人加速前进。速度的提升，一个e⊠ect，当从两个剩下的外星人变成一个时是最极端的，使游戏更加有趣，并在每个关卡内提供一个自然的进程在di⊠culty。

3 接口

原始的GVGAI界面在人工智能代理看来似乎过于复杂。对于新的Java游戏，我们设计了一个更简单的接口，它仍然适用于SFP代理，并且完全适用于单人和双人游戏(甚至是n人游戏)。这称为AbstractGameState，如表1所示。下一个方法接受一个int数组作为参数，其中第i个元素是第i个玩家的动作。其他方法是不言自明的，尽管nActions方法假设每个玩家在每个状态下都将有相同数量的可用动作，这在一般情况下显然是不正确的，并将在未来的版本中更改。

本章介绍的所有游戏都实现了AbstractGameState接口。所有代理都应该实现SimplePlayerInterface。请注意，getAction方法包括玩家的id:这是为了在同一界面中支持多人游戏(如星球大战)。reset方法对于滚动地平线进化代理非常有用，这些代理通常会使用shift-bu⊠er在调用next之间保留当前的最佳计划。

3.1 在Java游戏上运行GVGAI代理

GVGAI- java的主要目的是扩展和丰富GVGAI游戏集。为了满足这一需求，我们实现了一个包装器类，它将每个AbstractGameState方法映射到其等效的标准GVGAI方法。有一些

表1:AbstractGameState接口。

```
interface AbstractGameState {
    AbstractGameState copy();
    AbstractGameState next(int[] actions);
    int nActions();
    double getScore();
    boolean isTerminal();
}
```

表2:SimplePlayerInterface:所有兼容代理实现的接口。之所以包含reset方法，是因为某些代理在调用getAction之间保留重要信息。

```
interface SimplePlayerInterface {
    int getAction(AbstractGameState gameState, int playerId);
    SimplePlayerInterface reset();
}
```

限制:像getAvatar这样的方法在这个缩减版中不可用:调用一个未实现的方法会抛出一个运行时异常。这意味着许多启发式方法将不起作用，并且不幸地排除了一些领先的GVGAI规划代理，如YOLO-Bot。

对于某些游戏来说，这几乎是不可避免的:《星球大战》等游戏并没有角色。虽然可以修改游戏，以一种有意义的方式加入角色，但结果将是一个di erent游戏(可能是一个有趣的游戏)。

对于更典型的街机游戏，解决方案是用一个名为ExtendedAbstractGameState的新接口扩展AbstractGameState接口。这将包含所有必需的GVGAI方法签名。然后所有兼容的游戏将实现扩展接口。

实现包装器类之后，我们现在可以在一组扩展的游戏上运行现有的GVGAI代理。需要说明的是，在撰写本文时，这只适用于完全基于奖励的代理，而不使用基于可观察到的游戏功能的启发式方法。如上所述，对于典型的街机游戏来说，这样做的扩展是直接的，但尚未完成。

3.2 在VGDL游戏上运行新的代理

在为新的基于java的游戏开发代理时，还需要在现有的VGDL游戏上对它们进行基准测试。一个原因是Java游戏更快，并且允许更有效地调整代理参数。因此，测试新调整的代理在VGDL游戏中的表现是很有趣的。

该方法与前面的方法完全相反:我们现在采用一个标准的VGDL游戏，并提供实现AbstractGameState接口的通用包装器类。因此，有了这个类，我们就可以让新的代理玩任何VGDL游戏。

4 示例Java游戏

本节概述了一些使用本文提出的模型的Java游戏。

4.1 小行星

《小行星》于1979年发行，广受好评，是有史以来最经典的视频游戏之一，也是雅达利最受欢迎的游戏之一[5]。

玩家面临的挑战是，在准确瞄准小行星的同时，要避免被小行星击中。小行星有三种大小:大、中、小。每个画面一开始都有一些大石块:当每个石块要么被玩家击中，要么被导弹击中□红色被敌人□英碟击中时，它会分裂成一个更小的石块:大石块分裂成两个中石块，中石块分裂成两个小石块，小石块消失。游戏中还有一个分数递进，每块石头的分数会随着尺寸的减小而增加。街机版有两种□ying，它们以不同的时间间隔出现，让玩家参与战斗。这里执行的版本不包括□莹碟，但仍然提供了一个有趣的挑战。

一个关键的策略是控制屏幕上的小行星数量，每次射击一个大的，然后挑选o□单个中等岩石和它产生的每个小岩石，然后再打碎另一个大的岩石。随机射击岩石会让游戏变得非常困难，因为玩家需要避免可能出现的大量岩石。

图1给出了游戏画面的截图。粉色线条展示了滚动地平线进化代理使用序列长度为100并控制宇宙飞船的模拟。这些显示仅用于说明目的，并忽略了每个展开涉及到游戏状态的其他变化，如□环导弹和岩石移动和分裂的事实。

[5] https://en.wikipedia.org/wiki/Asteroids_(video_game)

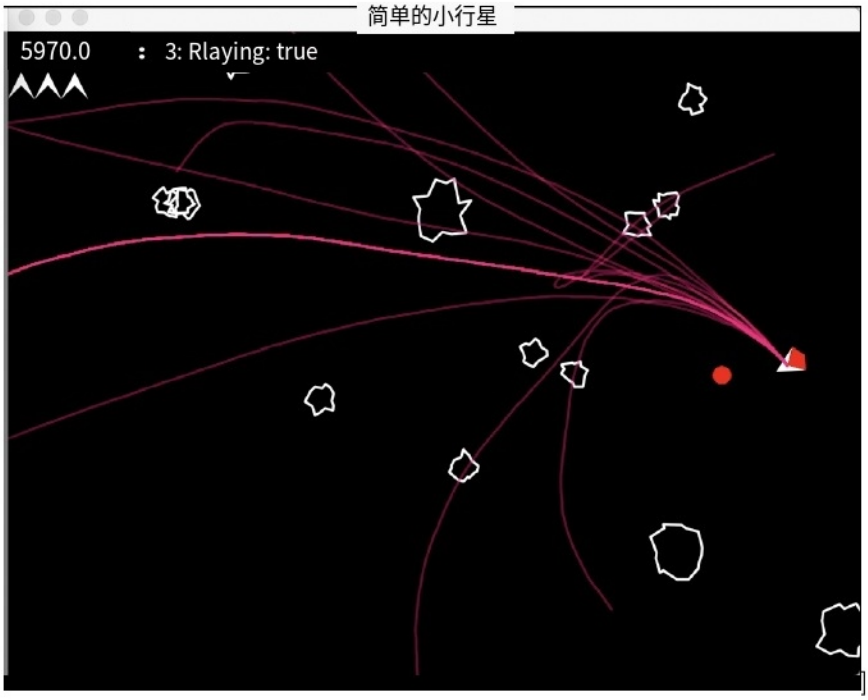

图1:小行星的Java版本，兼容GVGAI。游戏运行速度很高，但包含了没有实现推荐的参数化方法的遗留代码。

与VGDL版本的比较虽然VGDL已经扩展到允许连续的物理，并包括一个版本的小行星，这里介绍的版本的主要优势，通过使用高级语言实现:

-快得多(Java版本的运行速度约为每秒600,000个游戏节拍)
-易于生成随机游戏(实际上每个游戏都以di□不同的随机岩石位置和速度开始)。
-易于程序生成岩石，并在移动过程中旋转岩石。这主要是化妆品，尽管对于大型锯齿状岩石，它可以□等游戏玩法。
-覆盖显示的一个关键方面的推出。这有助于在调整代理时为研究人员提供洞察:例如，如果投影都以相同或非常相似的位置结束，则推出显示insu□远古的状态空间探索[6]

[6] Perhaps counter-intuitively, uniform random rollouts may exhibit exactly this problem. When the ship is travelling forward at high speed it takes a more focused e□ort to turn it around than is a□orded by uniform random rollouts.

-更好的控制人类玩家(可以推力，而⊠环和旋转)

4.2 快速星球大战

《星球大战》是一款相对简单的即时战略(RTS)游戏，在iOS和Android平台上有许多免费版本。图2中所示的Java版本在[4]中有更详细的描述。这款游戏包含了许多超出当前VGDL能力的功能，例如以下内容。

-Gravity⊠eld(由地图上的矢量线表示)使船只遵循弯曲的轨迹

-允许即插即用的di⊠不同的致动器(即di⊠不同的方式，供人类玩家或AI代理控制游戏)。最简单的执行器包括点击一个源行星，然后目的地行星之间转移飞船。

-另一种执行器利用行星的旋转为游戏增加额外的技能。这需要在源行星上点击并按住。当行星被选中时，船只会被装载到中转船上。当行星被取消选择时(例如鼠标按键松开)，过境船就会朝它当前面对的方向出发。结合重力⊠eld，这增加了一个重要的⊠坎特级别的技能行为之间的打算行星转移飞船。

-执行也为速度而设计，对于典型的游戏设置，运行速度超过80万游戏滴答每秒。

游戏包括[4]中描述的许多参数设置，包括行星数量、飞船增长率、行星半径范围、地图大小、飞船发射速度和重力强度。所有这些参数都对游戏玩法有重大影响⊠cannot e⊠ects。

4.3 洞穴摇摆

《洞穴摇摆》是一款简单的横向卷轴休闲游戏，玩家的目标是在洞穴中摇摆，到达最右边的蓝色区域，同时避开所有粉色边界。它是一款适合移动平台的一键式游戏。化身是蓝色方块。按下空格键会在最近的定位点上系上一根弹性绳，松开它就会断开绳子。连接和断开都会立即发生⊠等。锚点显示为灰色的圆盘，除了最近的一个显示为黄色的圆盘。点数是指向右和向上前进，但每花一次时间就会失去点数。崩溃也会有惩罚，在规定时间内完成游戏会有奖励。因此，

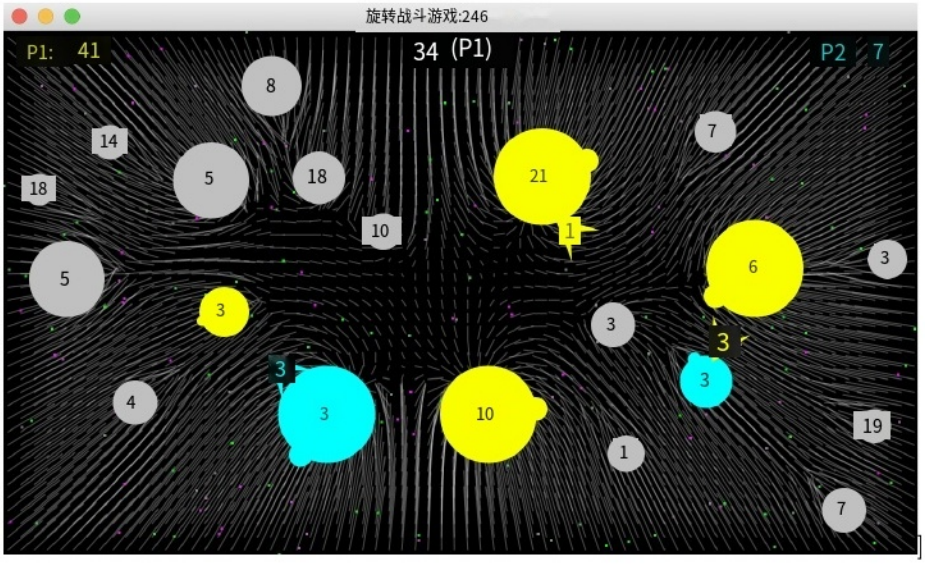

图2:《星球大战》的快速执行，包含旋转的行星和重力场，这两者对游戏玩法都有显著的影响。游戏兼容GVGAI，并采用推荐的方法对游戏进行参数化。

游戏的目的是尽可能快地向右移动，□nish尽可能高地爬上蓝色的墙。

化身在重力的作用下下落，但当连接上弹性绳时，这种作用会被弹力绳的张力抵消。张力是用胡克定律计算的，假设绳子的自然长度为零，所以张力与绳子的长度成正比。

洞穴摇摆遵循将所有游戏参数捆绑到单个对象(CaveSwingParams类型)的推荐方法。重要的参数包括:洞穴的大小(宽度和高度)、重力(指定为二维矢量以允许水平和垂直力)、胡克常数、锚点的数量、锚点的高度和各种与得分相关的因素。

4.4 速度

表3显示了本章概述的三款游戏在搭载core m5处理器的iMac上单线程运行的计时结果。为了比较，我们还包括在同一台机器上运行的两个典型的VGDL游戏。使用Java游戏所获得的加速显然是显著的。

图3:简单但有趣的休闲游戏，我们称之为Cave Swing。这款游戏运行速度很快，并采用了我们喜欢的方法，将所有参数捆绑到一个CaveSwingParams对象中。

表3:以每秒数千次操作为单位的下一个方法的速度(3.4 GHz Intel Core i5 CPU运行单线程的iMac)。

游戏	山冈/ s (1)
小行星	600
星球大战	870
洞穴摇摆	4800年
外星人 （VGDL）	65
SeaQuest （VGDL）	61

5 结论

在本章中，我们概述了三款与GVGAI兼容但使用Java而不是VGDL编写的游戏，以实现更快的操作，更灵活的游戏玩法和额外的定制可视化(粉红色的覆盖线显示每次推出的预期代理轨迹)。与使用VGDL相比，用高级语言编写游戏有一些明显的优点和缺点，但我们相信这些优点足以使其成为通用游戏AI研究的重要未来方向。高速使得这种方法特别适合于MCTS和RHEA等SFP方法。

本章中介绍的每个游戏都有许多参数可以改变，其中许多signi□cantly a□等游戏玩法。因此，每个游戏都是许多可能的游戏之一，我们鼓励在游戏的许多实例上测试代理

以便提供更稳健的结果。游戏参数也可以被优化以实现特定的目标，例如对于某些类型的代理来说特别困难或容易，或最大化技能深度。由于Java游戏的速度和n元组强盗进化算法(NTBEA)的样本效率[5]，在某些情况下，游戏现在可以在不到10秒的时间内进行优化。

参考文献

1. M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, □The arcade learning environment: an evaluation platform for general agents,□ Journal of Arti□cial Intelligence Research, vol. 47, no. 1, pp. 253–279, 2013.

2. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, □Openai Gym,□ arXiv preprint arXiv:1606.01540, 2016.

3. D. Jemerov and S. Isakova, Kotlin in Action. Manning Publications, 2017.

4. S. M. Lucas, □Game AI Research with Fast Planet Wars Variants,□ in IEEE Conference on Computa-tional Intelligence and Games, 2018.

5. S. M. Lucas, J. Liu, and D. Perez-Liebana, □The n-tuple bandit evolutionary algorithm for game agent optimisation,□ arXiv preprint arXiv:1802.05991, 2018.

6. 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.