第四章- GVGAI规划的前沿

Diego Perez-Liebana和Raluca D. Gaina

1 介绍

多项研究解决了GV-GAI规划轨迹中呈现的问题。本章旨在介绍GVGAI的最新技术，并在第2节中描述该领域最成功的方法。值得强调的是，这些方法是基于树搜索方法的不同变体，主要是MCTS(见前一章)，并添加了一些增强或启发式方法。这些提交给竞赛的智能体显示了玩多个游戏的能力，最大限度地减少了给定的游戏依赖启发式的数量，真正体现了GVGAI的精神。

观察和分析这些方法是很重要的，因为尽管是最亲⊠古老的方法，但它们在这个问题上仍远未达到最优性能。平均而言，这些方法在玩的游戏中取得了50%左右的胜率。通过研究这些方法的局限性，人们可以构建更强大的算法，更好地解决这一挑战。第3节总结了为了在这一领域取得进展必须面临的公开挑战。

最有希望的选择之一是自动从玩的游戏中提取信息，试图理解它。Di⊠不同的游戏可能会从不同的方法中受益⊠t，能够识别何时使用确定的算法或启发式可以提供一个巨大的飞跃。类似地，能够理解游戏中智能体的情况(即智能体是否在正确的轨道上获胜?)，可以提供关于何时改变游戏策略的自动指导。本章的第4节描述了我们从实时游戏玩法数据[8]预测游戏结果的方法。这意味着朝着组合方法又迈出了一步，该方法改善了一般视频游戏玩的规划挑战中的艺术结果的状态。

GVGAI计划是一个难题，我们已经达到了一般代理可以体面地玩一些可用游戏的地步。下一步可能是最难的(但可以说是最有趣的一个):我们能否自动分析游戏，并确定需要做什么才能获胜?我们如何理解任何游戏的目标，以及代理与实现这些目标的距离有多近?这些问题的答案可能会导致新一代的通用电子游戏代理。

2 GVGAI规划的最新技术

本节介绍了三个赢得了几个版本的GVGAI比赛的代理:OLETS, ToVo2和YOLOBOT。我们描述了实现的方法以及每种方法的优缺点。

2.1 OLETS (adrienctx)

开环期望值树搜索(OLETS)是由Adrien coueetoux为GVGAI开发的一种算法，该算法赢得了第六届单人GVGAI比赛。此外，它对2人规划赛道[4]的适配也成功赢得了这类赛道的两个版本(参见第2章)。

OLETS的灵感来自分层开环乐观规划(HOLOP)[20]，它使用前向模型(FM)以与MCTS类似的方式对动作进行采样，而无需将树的状态存储在内存中。虽然这不是非确定性领域的最优方法，但在实践中效果很好。与HOLOP的一个关键区别是，OLETS不使用默认的UCB策略进行操作选择，而是使用一种称为开环期望值(OLE)的新方法。OLE使用$r_M(n)$代替策略中奖励的平均值，这是两个组件的线性组合:从经过节点n的模拟计算的经验平均奖励及其子节点中的最大值r_M。另一个关键区别是OLETS不执行任何rollout。等式1显示了OLE分数，其中$n_s(n)$是访问状态s的次数，$n_s(a)$是从状态s中选择动作a的次数。

$$score = r_M(a) + \sqrt{\frac{ln(n_s(n))}{n_s(a)}} \tag{1}$$

算法1详细描述了OLETS。OLETS试图通过在状态值函数中添加禁忌偏见来奖励对关卡的探索。当新的节点被添加到树中，如果它之前少于T步之前被访问过，它们的状态会受到惩罚(小于1的数量级)。$10 < T < 30$的值已被证明提供了良好的实证结果。

OLETS的缺点之一是缺乏学习。这是一个非常简单和☒灵活的方法，它给了代理良好的反应能力，主要是由于开环搜索(与在节点中存储游戏状态的闭环相比)。然而，它在需要长时间计算来解决需要更深入规划的游戏的游戏中举步维艰。

算法1 OLETS，来自[13]。$n_s(n)$:经过节点n的模拟次数;$n_e(n)$:以e结尾的模拟次数;$R_e(n)$:累积奖励来自
simulations ended in n; $C(n)$: set of children of n; $P(n)$: parent of n.

Input: s: current state of the game.
Input: T: time budget.
Output: action: to play by the agent in the game.

```
1: function OLETS(s, T)
2:     T ←root initialize the tree
3:     while elapsed time < T do
4:         RunSimulation(T, s)
5:     return action = arg max_{a∈C(root)} n_s(a)
6:
7: function RunSimulation(T, s_0)
8:     s ← s_0 set initial state
9:     n ←root(T) start by pointing at the root
10:    Exit ←False
11:    while ¬Final(s) ∧ ¬Exit do
12:        if n has unexplored actions then
13:            a ←Random unexplored action
14:            s ←ForwardModel(s,a)
15:            n ←NewNode(a,Score(s))
16:            Exit ←True
17:        else
18:            a ← arg max_{a∈C(n)} OLE(n,a) select a branch with OLE (Equation 1)
19:            n ← a
20:            s ←ForwardModel(s,a)
21:    n_e(n) ← n_e(n) + 1
22:    R_e(n) ← R_e(n) + Score(s)
23:    while ¬P(n) = ∅ do
24:        n_s(n) ← n_s(n) + 1 update the tree
25:        r_M(n) ← R_e(n)/n_s(n) + (1−n_e(n))/n_s(n) max_{c∈C(n)} r_M(c)
26:        n ← P(n)
```

2.2 ToVo2

ToVo2控制器是Tom Vodopivec开发的智能体，灵感来自MCTS和强化学习[16]。ToVo2是这个双人GVGAI规划竞赛的第一个获胜者。

ToVo2通过将MCTS与Sarsa-UCT(λ)相结合对其进行增强[17]，将UCT的泛化与萨尔萨病的学习能力相结合。该算法也是一种开环方法，在MCTS的选择和仿真阶段使用FM计算状态-动作值。奖励被归一化到[0,1]来进行组合

萨尔萨病的UCB1政策。其他参数是探索率C = √2，奖励贴现率γ= 0.99和资格迹衰减率λ= 0.6。

模拟步骤计算访问的所有状态的值，而不是只观察在rollout结束时发现的状态。这个值被计算为两个连续的游戏时钟之间得分的di⊠erence。在一次迭代中所有访问过的节点都被保留，并且由于更新的步长参数，每次搜索后50%的知识被遗忘。对手被建模为随机均匀移动(即假定它是环境的一部分)。

MCTS仿真阶段的两个增强增强了控制器。⊠rst一种是加权随机部署，它偏向于行动选择，以促进探索，并减少再次访问相同状态的频率。第二种是动态展开长度，旨在初步探索头像的近处(从根部开始5次移动深度)，然后逐步增加深度(每5次迭代增加5次，最多50次)，以搜索更遥远的目标。

与《OLETS》类似，《ToVo2》也在努力应对那些在玩家附近看不到奖励的游戏，但它在处理发生在近视界的事件时却很强大。在希望探索关卡的游戏中，加权随机的推出被证明是有益的⊠社交，但如果o⊠在需要执行精确和特定移动序列的益智游戏中没有优势。

2.3 YOLOBOT

一个值得讨论的提交代理是YOLOBOT，它赢得了三个版本的GVGAI单人规划赛道(在其中表现优于OLETS)。YOLOBOT是由Tobias Joppen, Miriam Moneke和Nils Schr¨oder开发的，其完整描述可以在[10]中找到。

YOLOBOT是两种不同方法的组合:启发式引导的最佳优先搜索和MCTS。前者用于确定性博弈，而后者用于随机博弈。启发式搜索试图⊠nd一系列可以精确复制的动作。在开始时，代理返回NIL动作，直到找到该序列，达到游戏时间限制或发现游戏是随机的。在后一种情况下，代理切换到增强的MCTS来玩游戏。这个算法在几个方面得到了增强。

知情先验:YOLOBOT保持并动态更新一个知识库，预测游戏中的事件和运动。这个知识库在启发式中用于初始化未访问节点的UCT值，使MCTS通过那些似乎会导致更多的操作来扩展

有前途的职位。一旦MCTS从这些值开始模拟，这些值就会被忽略。

-知情的推出策略:在MCTS的模拟阶段，使用相同的启发式来确定和有希望的位置，以进行偏差行动选择。

-回溯:当☒发现一个失败的终端状态时，该算法回溯一个移动，并模拟多达四种替代行动。如果这些替代动作中的一个导致了一个不输的状态，那么这个动作的值就会反向传播。

-搜索空间的修剪:FM用于确定应用动作a时s的下一个状态是否与NIL播放时达到的状态相同。在这种情况下，所有满足这个条件的动作a都会被剪枝。其他导致角色超出等级界限的动作，以及导致游戏状态失败的动作也会被修剪。

自成立以来，YOLOBOT在GVGAI比赛中取得了最高的胜率。这种智能体在确定性游戏(谜题)中工作得很好，在这种游戏中，解决方案可以在短到中长动作序列中到达，在玩随机游戏时，它的表现优于其他方法。然而，这并不意味着YOLOBOT在所有游戏中都表现出色。如第2节所示，YOLOBOT在该框架的不同游戏集中取得了41.6%至63.8%的胜率。仍然有大约50%的游戏胜利逃脱，即使是为这个挑战而创造的最强的控制器之一。

3 当前GVGAI规划中的问题

GVGAI的两个规划轨道(单人和双人)自框架建立和公开以来受到了最多的关注。正如上一节所暗示的和第2章所示，最好的方法并没有达到高于50%的胜率。此外，许多游戏都是在非常罕见的情况下解决的，不同的MCTS和RHEA变体在框架的所有(超过100个)游戏中努力获得超过25%的胜利。因此，目前的主要挑战之一就是在框架的所有游戏中提高胜率。

最近一项关于GVGAI方法的调查[14]描述了许多试图解决这个问题的算法的增强。在大多数情况下，包括本书中描述的那些，这些改进确实实现了提高所尝试的游戏子集的性能，但通常有另一个子集的性能下降或(在最好的情况下)保持在相同的水平。GVGP的本质让这一点变得可以理解，因为很难想出适用于所有游戏的方法——但这并不意味着这不是一个需要解决的问题。

然而，最近的调查显示，一些工作可以提供显着□不能在未来的进步。例如，如果使用更复杂的度量(不仅是直接的a *，还包括其他方法，如potential□elds[3])，那么从精灵中提取特征的效果似乎比简单的欧几里得距离要好。如何将这种更复杂的计算与游戏的实时性结合起来，特别是在那些明智地使用预算至关重要的游戏中，这仍然是未来研究的问题。

明智地使用预算时间是一个重要的改进点。有大量的方法试图在思考时间使用FM访问的状态来了解比赛并提取特征以偏向进一步的搜索(如[12]和[10])。在大多数情况下，这些方法工作得很好，在整体情况下提供了边际改进，但这些特征仍然是针对一组游戏量身定制的，缺乏通用性。简而言之，当研究人员设计特征提取器时，他们(自然)会受到他们所了解的游戏以及其中重要的内容的影响□，但其他一些游戏可能需要更复杂或从未想过的功能。自动通用特征提取器的设计是GVGP最大的挑战之一。

另一个有趣的方法是在动作空间上工作，以使用更抽象的动作。这可以采用宏操作(作为一个整体处理的原子操作序列[15])或选项(在MCTS[19]中，与目标相关)的方法。这种方法使动作空间更粗(减少了几个连续回合的动作空间)，并最大化使用预算时间:一旦一个宏观动作开始执行(需要T时间步到□nish)，控制器可以计划下一个宏观动作，依靠T−1时间步来完成搜索。对于需要长期规划的游戏来说，这可能是一个有趣的方法，这是一个游戏子集，对于目前的方法来说是困难的。使用这些动作抽象的结果再次表明，它们在某些游戏中有帮助，但在其他游戏中却没有。他们还表明，有些游戏适合□t from di□不同长度的宏观动作集合，而其他游戏则更适合与其他游戏一起玩。

我们可以合理地认为，鉴于某些算法在某些游戏中的表现比其他游戏更好，并且有些游戏通过di□erent方法玩得更好，一种试图自动确定什么是适合正确游戏的正确算法的方法应该会有很大的帮助。事实上，已经讨论过的YOLOBOT代理在这方面进行了□rst的尝试，通过区分确定性游戏和随机游戏。游戏classi□阳离子[11][1]以及混合或超启发式方法的使用，实际上是一个有趣的研究领域。我们的目标将是建立一个classi□er，动态地确定在当前游戏中使用哪种算法是最好的，然后切换到它。已经有一些尝试对游戏进行分类使用

提取的特征，尽管最新的结果似乎表明这些classi⊠阳离子(以及使用的算法)不够强，不足以在许多游戏中表现良好。

我们在下一章开始探索的一个有趣的方向是使用更通用的特征，专注于智能体如何体验游戏，而不是它固有的特征(因此有偏差)。接下来的研究将⊠rst砖块放在一个系统中，该系统设计用于仅基于智能体游戏功能在游戏中的算法之间切换。特别是，下一节描述了如何构建一个获胜预测器，以根据agent经验确定游戏最可能的结果。

4 GVGAI中的一般获胜预测

这项工作的目的是建立一个仅基于智能体在玩任何GVGAI游戏时的经验测量的结果预测器。我们有意将与游戏相关的功能(如npc或资源的存在)留在这项研究之外，这样这些见解就可以转移到其他框架或特定游戏中，而无需太多改变。最重要的要求是游戏依赖于前向模型(Forward Model, FM)、游戏分数和带有输赢结果的游戏结束状态。

4.1 游戏玩代理和功能

建立这些预测器的⊠rst步骤是收集训练预测模型所需的数据。这些数据将从玩GVGAI游戏的代理中检索。使用的算法如下。

随机搜索(Random Search, RS) RS代理随机抽取长度为L的动作序列，直到分配的预算耗尽。然后，在游戏中播放最佳序列的⊠rst动作。使用FM来评估序列，以应用当前游戏状态中的所有动作，并根据公式2为最终状态分配一个值。H^+是一个很大的正整数(而H -是一个很大的负数)。

$$f = score + \begin{cases} H^+, \ if \ loss \\ H^-, \ if \ win \end{cases} \tag{2}$$

根据L: 10、30和90的值，本研究中使用了三种RS参数。

滚动地平线进化算法(RHEA)在本研究中使用了RHEA代理(在第2章中解释)，使用了一些在以前的研究中表现良好的改进。所选的算法con⊠gurations如下:

-Vanilla RHEA[5]，算法的默认配置。
-EA-MCTS[6]，其中原始种群由MCTS播种。
-EA-Shift[7]，其中RHEA在个体结束时使用Shift和蒙特卡罗滚动来增强。
-EA-All，它结合了EA-Shift和EA-MCTS，以确保完整性。

对于这四种变体，使用的是两个di⊠不同的参数集:P=2, L=8和P=10, L=14，其中L是个体长度和P种群大小。使用公式2评估最终状态。

蒙特卡罗树搜索(MCTS)使用香草MCTS算法，再次使用公式2来评估在部署结束时达到的状态。MCTS采用3个参数集:W=2, L=8;W=10, L=10和W=10, L=14。W是MCTS迭代的次数，L是从根状态开始的rollout的深度。

所有这14种算法在GVGAI框架的100个公共游戏中运行，每个游戏使用5个级别，每个级别重复20次。所有的算法在每场比赛中都有相同的预算来做决定:900次调用FM的advance功能。表1总结了这些方法在测试的100场游戏中得到的结果。

这些运行中的每一个都产生了两个日志⊠les，其中包含每个游戏状态下的agent和游戏状态信息。关于游戏，每一步的分数和⊠nal游戏结果(赢/输)被保存。关于代理，我们记录了在每个游戏步骤中进行的操作和如下所述的一组功能。

-ϕ_1当前游戏得分。
-ϕ_2收敛性:算法找到推荐的⊠nal解时的迭代次数，直到进化结束时才再次改变，在一个游戏步骤期间。低值表示快速且几乎随机的决策。
-ϕ_3积极奖励:积极得分事件的计数。
-ϕ_4Negative rewards:消极评分事件的计数。
-ϕ_5成功:一条线在所有获胜计数上的斜率。这个计数反映了在搜索过程中，在任何时间点上以获胜告终的状态数⊠。高值表示随着游戏的进行，获胜状态的发现增加。

	算法	胜率(标准误差)
1	10-14-EA-Shift	26.02 % (2.11)
2	2-8-EA-Shift	24.54 % (2.00)
3	10-RS	24.33 % (2.13)
4	14-MCTS	24.29 % (1.74)
5	10-MCTS	24.01 % (1.65)
6	10-14-EA-MCTS	23.99 % (1.80)
7	2-8-EA-MCTS	23.98 % (1.73)
8	2-8-EA-All	23.95 % (1.98)
9	8-MCTS	23.42 % (1.61)
10	10-14-RHEA	23.23 % (2.08)
11	10-14-EA-All	22.66 % (2.02)
12	30-RS	22.49 % (2.02)
13	2-8-RHEA	18.33 % (1.77)
14	90 - rs	16.31 % (1.67)

表1:本研究在100场GVGAI游戏中使用的所有方法的胜率(和标准误差)。类型和配置☒配置(rollout length L如果一个值，人口规模P和rollout length L如果两个值)被报告。

-ϕ_6危险:一条线在所有损失计数上的斜率。在搜索过程中，每发现一个终端游戏损失，这个数值就会增长。高值表示随着游戏的进行，丢失状态的数量会增加。

-ϕ_7改进:给定自游戏开始以来看到的最佳☒tness值，改进是a这个增量对游戏tick的斜率。高值表示最佳☒tness值随着游戏的进行而增加。

-ϕ_8决断性:这是Shannon熵(SE，见公式3)关于每个动作被推荐的次数。在所有将特征计算为SE的情况下，高值表明具有相似值的操作;相反的则显示这些动作中的一些被推荐的频率更高。

-ϕ_9选项探索:每个可能的操作被探索的次数的SE。在这种情况下，这反映了☒在搜索过程中，在任何时间这个行动是一个解决方案的☒rst移动了多少次。低的值表明探索的行动中的不平衡，而高的值意味着在搜索过程中所有行动的探索大致相同☒rst移动。

-ϕ_{10}适应度分布:每个动作的SE比。

-ϕ_{11}成功分布:SE大于每次行动的胜利数。

-ϕ_{12}危险分布:SE除以每次行动的损失计数。

$$H(X) = -\sum_{i=0}^{N-1} p_i log_2 p_i \tag{3}$$

特征φ´´，φ´´，φ´´，$_{10}$，ϕ_{11}和φ´´计算从游戏开始到当前时间点的平均值。特征φ，φ´´，φ´´和φ´´依赖于FM。数据集和处理脚本已经公开[1]。

此外，功能分为3 di☒不同的游戏阶段:早期(☒rst游戏30%的ticks)，后期(游戏最后70%的ticks)和中期(游戏30% - 70%之间的ticks)。这些划分用于训练di☒erent阶段模型:早期，中期和后期游戏classi☒ers。

特征相关性在di☒erent游戏阶段中的划分对应的信念是，di☒erent事件一般发生在游戏的开始和结束。因此，在di☒不同的游戏阶段训练模型可能会产生有趣的结果。图1显示了使用Pearson correlation coe☒cient的特征之间的相关性。这比较了早期(左)和后期(右)的游戏特征，显示了小(但存在)的di☒erences。例如，位于游戏后期右下角的特征之间的相关性要高于游戏前期。

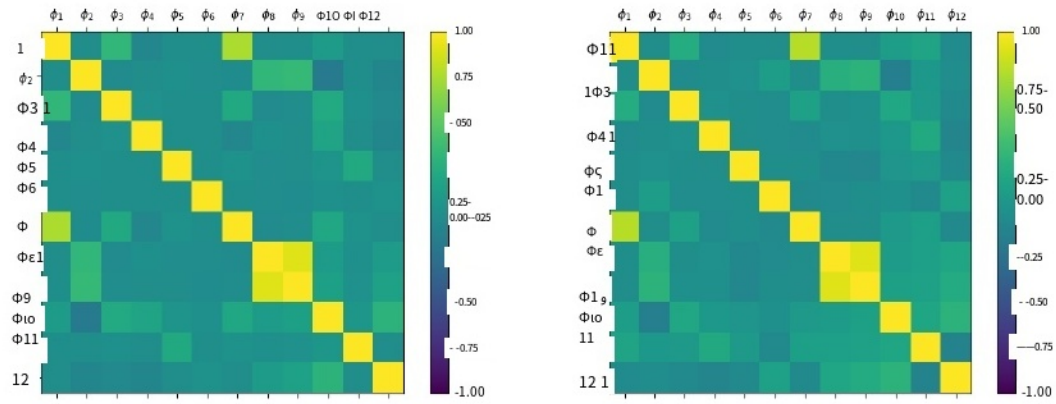

图1:游戏前期(左，所有游戏的0-30%)和游戏后期(右，所有游戏的70-100%)的特征相关性

[1] https://github. com/rdgain/ExperimentData/tree/GeneralWinPred-CIG-18. The ☒nal data is split over 281, 000 ☒les. It took approximately 2.5 hours to generate this database from raw data ☒les, using a Dell Windows 10 PC, 3.4 GHz, Intel Core i7, 16GB RAM, 4 cores.

游戏后期阶段也显示出收敛(ϕ_2)和危险(ϕ_6)之间有趣的正相关关系。这可能表明，当在搜索范围内发现更多可能的损失时，代理需要更多的时间来决定他们的⊠nal决策。在游戏后期不那么强的正相关是⊠tness改善和⊠tness分布之间的行动，这意味着当一个行动被认为明显⊠优于其他行动时，⊠tness不太可能进一步改善，可能是由于其他行动没有被进一步探索。最后，收敛性和⊠tness分布之间存在持续的负相关关系。这似乎表明，当⊠发现一个被认为明显优于其他行为的行动时，一个智能体不太可能改变其决定。

4.2 预测模型

对于为本研究训练的所有模型，数据被随机划分为训练集(80个数据)和测试集(20个)。使用描述为输入的特征和作为标签和预测的输赢来构建几个classi⊠er模型。在本节中，根据精度、召回率和F1分数(分别为公式4、公式5和公式6)报告模型预测质量:

$$precision = \frac{TP}{TP + FP} \tag{4}$$

$$recall = \frac{TP}{TP + FN} \tag{5}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{6}$$

TP代表真阳性(正确预测获胜)，FP代表假阳性(错误预测获胜)，FN代表假阴性(错误预测失败)。由于一般代理的整体表现较低(所有游戏的胜率接近25%)和GVGAI游戏的多样性高，F1分数是报告的主要指标。

基线模型为了判断训练出的模型是否优于简单的专家规则系统，建立了一个基线模型。在大多数游戏(游戏邦注:如GVGAI和其他街机游戏)中，通过游戏获得分数通常是一种很好的进程指示器,

因此，它通常会带来胜利。方程7是一个简单的模型，它比较了正负分数事件的数量。

$$\hat{y} = \begin{cases} win & if\ \phi_3 > \phi_4 \\ lose & otherwise \end{cases} \tag{7}$$

这个classi☒er在测试集上的表现可以在表2中看到。可以观察到，尽管具有很高的精度(0.70)，但它的F1-Score为0.59。本章剩余部分将该模型称为R_g。

	精度	回忆	F1-Score	支持
损失	0.83	0.52	0.64	20500
赢得	0.35	0.70	0.46	7500
Avg / 总计	0.70	0.57	0.59	28000

表2:基于全局规则的classi☒er报告。在测试集中所有实例的所有游戏tick上测试的全局模型。

Classi☒er选择-全局模型在这项工作中，我们训练和测试了7个Classi☒ers，作为win预测任务的概念证明。这些分类是k近邻(5个邻居)，决策树(5个最大深度)，随机森林(5个最大深度，10个估计器)，多层感知器(学习率1)，AdaBoost-SAMME[21]，朴素贝叶斯和Dummy(一个简单的规则系统，也用作另一个基线)。所有类都使用Scikit-Learn Python 2.7.14库中的实现[2]。上述未指定的参数在Scikit-Learn库中设置为各自的默认值。

使用交叉验证(10折)来评估性能。classi☒ers得到的，按照上面相同的顺序，在评估期间的准确率为0.95,1.00,0.98,0.96,1.00,0.95和0.66。AdaBoost和决策树在验证和测试中获得了非常高的精度值(见表3)。其余的实验使用AdaBoost作为主要的分类模型(没有特别的理由超过决策树)。

表4显示了根据AdaBoost的teach特性的重要性。不出所料，游戏分数似乎是区分胜负最重要的特征。紧随其后的是代理看到的获胜次数、☒tness的提升和危险的衡量。根据训练的模型，智能体的果断似乎对决定比赛结果没有影响。

	精度	回忆	F1-Score	支持
损失	1.00	0.99	0.99	20500
赢得	0.97	0.99	0.98	7500
Avg / 总计	0.99	0.99	0.99	28000

表3:AdaBoost测试了测试集中所有实例的所有游戏节拍。

1	0.24	2	0.04	ϕ3	0.08	Φ0.06 4
Φ50.2		ϕ6	0.1		0.12	08 0
ϕ9	0.06	Φ10	0.02	Φ11	0.02	Φ0.06 12

表4:从全局模型中提取的特征的重要性。

模型训练预测在游戏期间的三个di☒不同级别上进行:游戏早期(☒rst 30%的游戏滴答)，游戏中期(30−70%)和游戏后期(后30%)。使用游戏中每个间隔对应的游戏节拍中捕获的特征对模型进行训练。在本节的其余部分中，这三个模型分别被称为E_g、M_g和L_g。

训练好的模型通过检查它们在20个测试游戏上的表现来进行测试(也考虑到游戏的tick interval)。10折交叉验证的训练分别提供了0.80、0.82和0.99作为游戏早期、中期和后期预测的结果。测试准确率分别为0.73、0.80和0.99,f1 - score分别为0.70、0.80和0.99。

现场比赛结果我们的下一步是模拟训练好的模型如何现场预测比赛结果(当代理在比赛时)。为此，我们通过提取log☒les中代理记录的游戏tick范围(T ={100·a:∀a∈[1,20]:a∈N})的特征来模拟游戏，这些特征都是从游戏开始到测试的当前tick T∈T为止。我们在20个测试游戏上采用了上述14种算法所玩的游戏，在它们的5个关卡中的每个关卡上玩了20次。每个模型被要求每100个滴答预测一次游戏结果。

图2和3显示了从这个测试中获得的结果。基于规则的基线模型在某些游戏中取得了很高的表现，比经过训练的预测模型表现得更好(例如，参见《外星人》、《Defem》、《Chopper》和《蛋癖》)。这些游戏有大量的得分事件，因此R_g的简单逻辑在这些情况下表现良好也就不足为奇了。然而，在其他游戏中，训练模型的预测效果比基线要好得多(参见《Ghost Buster》、《颜色逃脱》或《青蛙》)，这些游戏的结果与游戏分数没有显著相关性，简单的基于规则的预测也不准确。

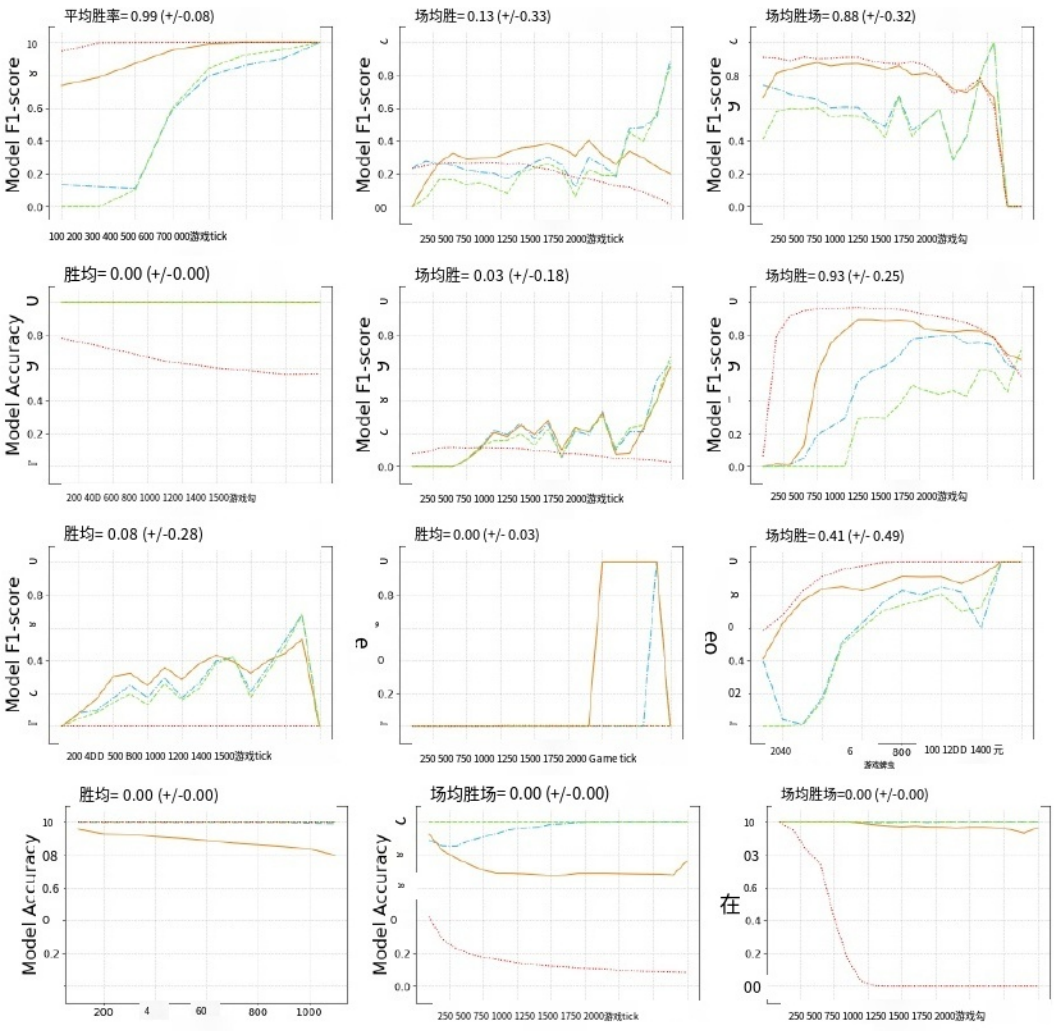

图2:测试集中每场比赛的模型f1得分，平均1400次，14个代理，每场比赛100次。游戏计时显示在X轴上，最多2000个游戏计时。三个di erent预测模型在游戏前期、中期和后期的数据特征上进行训练，以及基于基线规则的预测器。如果所有模型的f1得分都为0，则绘制精度图。此外，还报告了每场比赛的获胜平均值。游戏从上到下，从左到右:外星人，破石，黄油，蛋糕，追逐，直升机，颜色逃脱，警察，Defem, De De ection, DigDug和大金刚。

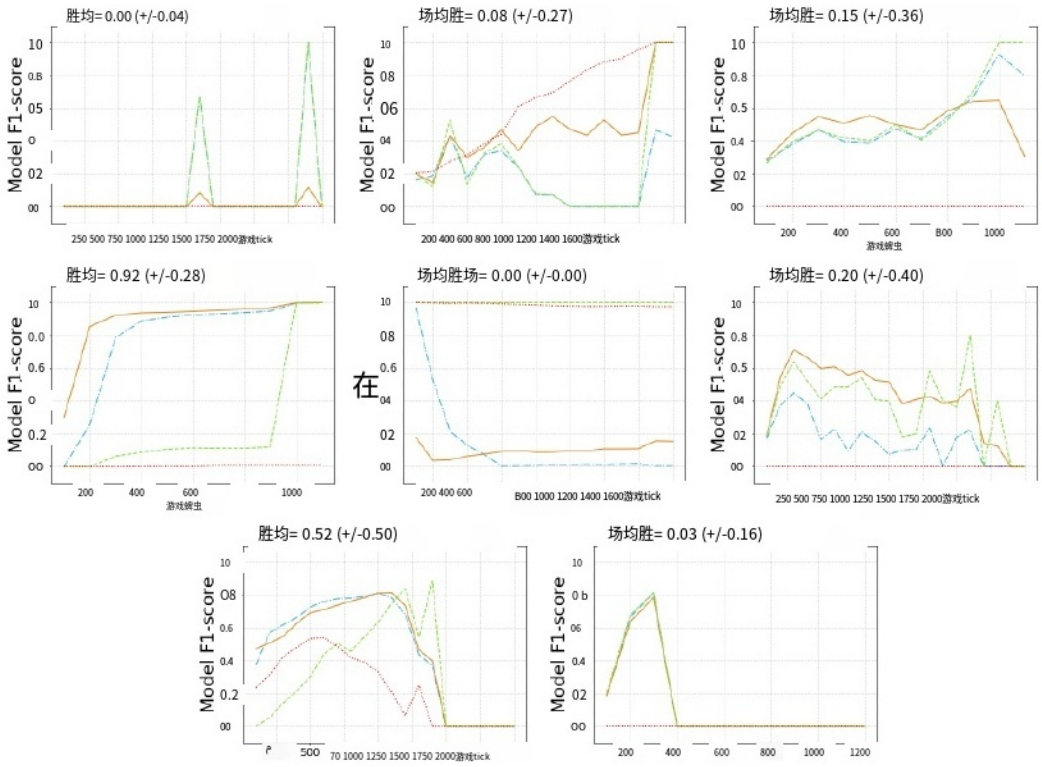

图3:测试集中每场比赛的模型f1得分，平均1400次，14个代理，每场比赛100次。游戏计时显示在X轴上，最多2000个游戏计时。三个di⊠erent预测模型在游戏前期、中期和后期的数据特征上进行训练，以及基于基线规则的预测器。如果所有模型的f1得分都为0，则绘制精度图。此外，还报告了每场比赛的获胜平均值。游戏从上到下，从左到右依次为:Dungeon, Eggomania, Escape, Factory Manager, Fireman，青蛙，Ghost Buster和Hungry Birds。

值得一提的是，经过训练的模型并没有遵循预期的曲线。人们可以期待E_g在游戏早期表现得更好，然后在游戏进行时降低其准确性。M_g可以在游戏中期表现出很高的表现，L_go⊠只有在游戏结束时才会做出很好的预测。然而，早期的游戏预测器与其他预测器相比表现最差(这可以解释为缺乏此模型的可用信息)。后期的游戏模型在胜率非常低的游戏中是非常准确的(例如在消防员中，E_g和M_g是预测胜率，但总体胜率在这款游戏中仍然是0%)。

高f1得分指数表明预测者能够正确判断输赢。因此，关注那些胜率接近50%的游戏(如《Defem》和《Ghost Buster》)是很有趣的。在这种情况下，f1得分超过0.8的玩家只通过了一半的游戏。中间的模型M_g在这种情况下提供了非常好的结果，成为在这种情况下最好的预测器，也可能是最好的预测器。

值得注意的是，该模型能够在游戏进行到一半时(有时只是在游戏进行了四分之一之后)预测游戏的结果，即使游戏获胜或输掉的概率是一个硬币⊠ip。这些模型是通用的:它们已经在di⊠erent游戏中进行了训练，没有转播依赖于游戏的特征——只是代理体验测量。

因此，作为超启发式系统的一部分，使用这些预测器的范围很大。在本研究中测试的一些算法确实在胜率为50%的游戏中获胜，如Defem或Ghost Buster，并且找到一种方法为每个游戏使用适当的方法将提高GVGAI的性能。这样的系统将需要依靠一个准确的获胜预测模型(知道是否切换到一个di⊠erent方法是必需的)和第二个模型，确定哪个是最好的方法给定的特征观察(知道要改变到什么)。

表5总结了三种模型在所有游戏中不同游戏阶段的f1得分。该表显示的结果表明，基于规则的模型在所有游戏阶段提供了一致的表现。其早期表现较好(f1得分为0.42)。但在游戏中后期，M_g的f1得分分别为0.57分和0.71分，明显优于其他游戏。在所有的游戏和阶段中，中间游戏M_g模型是最好的，平均f1得分为0.53。M_g是个体中后期最强的模型，只有在游戏早期阶段被简单的规则预测器(它考虑到得分会导致胜利，因此纳入人类知识)所克服。

	Early-P	Mid-P	Late-P	Total-M
如	0.22 (0.72)	0.42 (0.74)	0.49 (0.76)	0.38 (0.74)
毫克	0.29 (0.72)	0.57 (0.79)	0.71 (0.83)	0.53 (0.78)
Lg	0.01 (0.73)	0.05 (0.74)	0.22 (0.76)	0.09 (0.74)
Rg	0.42 (0.67)	0.47 (0.61)	0.46 (0.58)	0.45 (0.62)
总磷	0.24 (0.71)	0.38 (0.72)	0.47 (0.73)	

表5:f1 -在所有游戏中，每个游戏阶段对每个模型进行评分，括号内为准确性。每一列是一个游戏阶段，每一行是一个模型。加粗突出显示的是每个游戏阶段的最佳模型，以及整体最佳阶段和模型。

最后，为了测试预测的稳健性，我们在使用RHEA训练的模型的同时，用MCTS对测试游戏进行了测试。从MCTS智能体中提取的智能体经验特征被输入到用不同算法训练的预测模型中。图4显示了这次测试和上一次测试的对比。左边是MCTS代理正在使用的模型。右边是RHEA和RS变异。可以看到，所有模型在游戏的di⊠不同阶段的表现都是相似的，并且能够在游戏中途准确预测游戏结果。

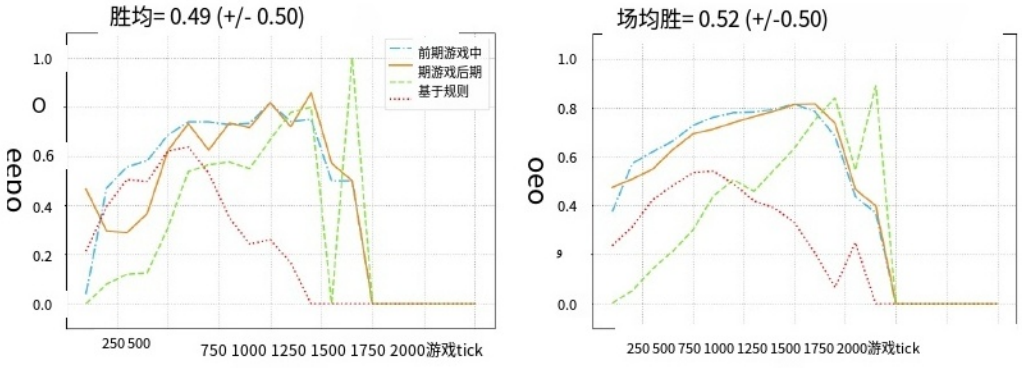

图4:使用RHEA和RS训练(80个训练游戏)和MCTS(左)和RHEA(右)测试(20个测试游戏)在《Ghost Buster》游戏中获得的f1分数。

接下来的步骤如前所述，合乎逻辑的下一步将是构建一个超启发式GVGAI代理，在玩任何游戏时，根据预测，它可以在算法之间切换。识别哪个是要切换到的算法可以分解为两个子任务:其中一个信号传递哪些特征测量需要改变，另一个识别哪个智能体可以交付所需的新行为。

关于⊠rst任务，可以分析⊠中的di⊠不同的特征如何影响每个任务。图5显示了《青蛙》(关卡0，由2-8-RHEA玩)中游戏tick 300时三个模型给出的预测示例。可以看到，每个模型的di⊠erent特征指标都被突出显示，表明哪些特征负责赢或输的预测。利用这种分析的超启发式方法有可能确定模型预测的原因。

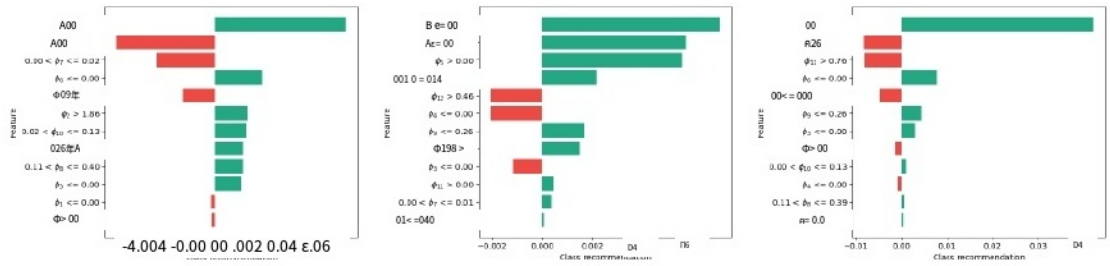

图5:基于特征的类预测(LIMEsystem[2]).)红色signi□es模型特征推荐输，绿色推荐赢。x轴上绘制了基于个体特征推荐的类别被选择的概率。

最后，该模型可以通过深度变体和更多特征来增强，例如赋能[9]、空间熵或表征智能体环境[18]。

参考文献

1. P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, □Matching Games and Algorithms for General Video Game Playing,□ in Twelfth Arti□cial Intelligence and Interactive Digital Entertainment Confer-ence, 2016.
2. L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, □API design for machine learning software: experiences from the scikit-learn project,□ in ECML PKDD Workshop: Languages for Data Mining and Machine Learning, 2013, pp. 108–122.
3. C. Y. Chu, T. Harada, and R. Thawonmas, □Biasing Monte-Carlo Rollouts with Potential Field in General Video Game Playing,□ , pp. 1–6, 2015.
4. 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.
5. R. D. Gaina, J. Liu, S. M. Lucas, and D. P´erez-Li´ebana, □Analysis of Vanilla Rolling Horizon Evo-lution Parameters in General Video Game Playing,□ in European Conference on the Applications of Evolutionary Computation. Springer, 2017, pp. 418–434.
6. R. D. Gaina, S. M. Lucas, and D. P´erez-Li´ebana, □Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing,□ in Conference on Evolutionary Computation. IEEE, 2017.
7. ——, □Rolling Horizon Evolution Enhancements in General Video Game Playing,□ in 2017 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2017.
8. R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, □General Win Prediction from Agent Experience,□ in Computational Intelligence and Games (CIG), IEEE Conference on, 2018.
9. C. Guckelsberger, C. Salge, J. Gow, and P. Cairns, □Predicting player experience without the player.: An exploratory study,□ in Proceedings of the Annual Symposium on Computer-Human Interaction in Play, ser. CHI PLAY ’17. New York, NY, USA: ACM, 2017, pp. 305–315.

[2] https://github.com/marcotcr/lime

10. T. Joppen, M. U. Moneke, N. Schr¨ oder, C. Wirth, and J. Furnkranz, ¨ ☒Informed Hybrid Game Tree Search for General Video Game Playing,☒ IEEE Transactions on Games, vol. 10, no. 1, pp. 78–90, 2018.

11. M. J. Nelson, ☒Investigating Vanilla MCTS Scaling on the GVG-AI Game Corpus,☒ in 2016 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2016, pp. 402–408.

12. D. Perez, S. Samothrakis, and S. Lucas, ☒Knowledge-based Fast Evolutionary MCTS for General Video Game Playing,☒ in Conference on Computational Intelligence and Games. IEEE, 2014, pp. 1–8.

13. D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Cou¨etoux, J. Lee, C.-U. Lim, and T. Thompson, ☒ The 2014 General Video Game Playing Competition,☒ IEEE Transactions on Compu-tational Intelligence and AI in Games, vol. 8, pp. 229–243, 2015.

14. D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, ☒General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms,☒ arXiv preprint arXiv: 1802.10363, 2018.

15. 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.

16. R. S. Sutton and A. G. Barto, Reinforcement Learning : An Introduction. MIT Press, 1998.

17. T. Vodopivec, S. Samothrakis, and B. Ster, ☒On monte carlo tree search and reinforcement learning,☒ Journal of Arti☒cial Intelligence Research, vol. 60, pp. 881–936, 2017.

18. V. Volz, D. Ashlock, S. Colton, S. Dahlskog, J. Liu, S. M. Lucas, D. P. Liebana, and T. Thompson, ☒ Gameplay Evaluation Measures,☒ in Articial and Computational Intelligence in Games: AI-Driven Game Design (Dagstuhl Seminar 17471), E. Andr´e, M. Cook, M. Preuß, and P. Spronck, Eds. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 36–39.

19. M. d. Waard, D. M. Roijers, and S. C. Bakkes, ☒Monte Carlo Tree Search with Options for General Video Game Playing,☒ in 2016 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2016, pp. 47–54.

20. A. Weinstein and M. L. Littman, ☒Bandit-Based Planning and Learning in Continuous-Action Markov Decision Processes,☒ in Proceedings of the Twenty-Second International Conference on Automated Plan-ning and Scheduling, ICAPS, Brazil, 2012.

21. J. Zhu, S. Rosset, H. Zou, and T. Hastie, ☒Multi-class AdaBoost,☒ Ann Arbor, vol. 1001, no. 48109, p. 1612, 2006.