Defeat Most Human: 现代规则的俄罗斯方块游戏 AI

陈益漳 成昂 黄程宇 傅欣毅

Jan 10th, 2020

摘要

我们使用 Approximate Q-Learning 算法训练 AI 挑战基于现代规则的俄罗斯方块，以击败大多数的人类玩家作为目的。将玩家（AI）可以获取到的所有关于游戏环境的信息，包括以二维数组表示的当前的游戏区域、当前的连击数量、分数、正在下落的方块、hold 的方块、未来的方块等信息视为 State，将游戏发出的方块放稳在游戏区域的某个位置这一操作视为 Action。同时我我们使用线性权重方程和四阶多项式权重方程作为评估函数，并比较两者表现。我们通过随机发送垃圾块来模拟对战，在经过大量的训练后，我们的 AI 已经可以坚持极长的时间。我们接下来预计让我们的 AI 能预测更多的方块，能够有计划得做出更复杂的攻击行为。

1 俄罗斯方块的历史

俄罗斯方块是世界上最著名的电子游戏之一，有着长达 35 年的历史。时至今日，在加入了以对战为重点的现代版规则后，俄罗斯方块仍然有许多爱好者。

作为一款久负盛名的游戏，俄罗斯方块一直是计算机科学家们研究的对象。现在，传统俄罗斯方块已经有了许多用 AI 来游玩的解决方案，而我们则想要用 AI 挑战基于现代规则的俄罗斯方块。

在现代规则下，玩家不仅需要努力消块来确保自己不死，还需要争取通过 t-spin、四消等方式给对手发送垃圾块，以此来攻击对手，以期获得最终的胜利。

任天堂开发的基于现代规则的游戏《Tetris99》，在这个游戏中 99 个玩家将同时游玩基于现代规则的俄罗斯方块，在保证自己不失败的前提下互相攻击，并最终决出一个胜者。由于不同的基于现代规则的俄罗斯方块游戏在一些细微的规则上都可能会有一定的区别，我们对现代规则的讨论将基于《Tetris99》的规则，以避免产生歧义。

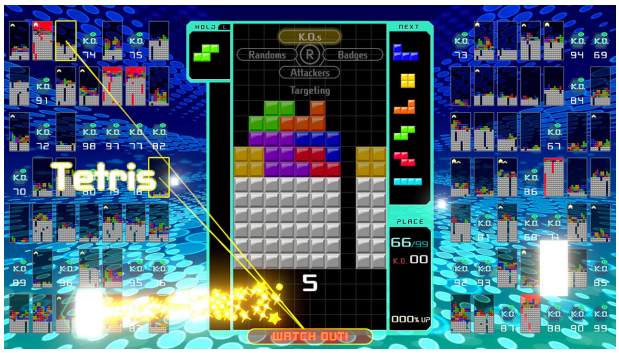

图 1: Tetris99

2 游戏规则

俄罗斯方块的基本规则是移动、旋转和摆放游戏自动输出的各种方块（S, Z, J, L, T, O, I），使之排列成完整的一行或多行从而消除。游戏失败的条件是方块堆积地过高以至于覆盖住了某一片区域（往往是方块的出生位置）。游戏的基本宗旨是通过积极的消行来避免游戏失败，并且在此基础上发展出了经典规则下的计分玩法以及现代规则下的对战玩法等玩法。

2.1 游戏目标

在传统规则的俄罗斯方块下，玩家的游戏目标是在保持游戏不失败的前提下获得较高的分数。游戏的得分方式基本上是通过消行，而消行所得的分数与一次性消除的行数是呈指数关系的，既一次性消行数每增加 1，此次消行的得分将乘以 2，因此在相同的消行总数下，进行的消四操作越多总分越高。传统俄罗斯方块会随着玩家消行总数的增长增加方块的下落速度，所以为了能够在游戏难度较低的时候获得尽量多的分数，玩家往往会倾向于只使用消四来进行消行，同时使游戏场地中的方块排列尽量规整、高度尽量低。

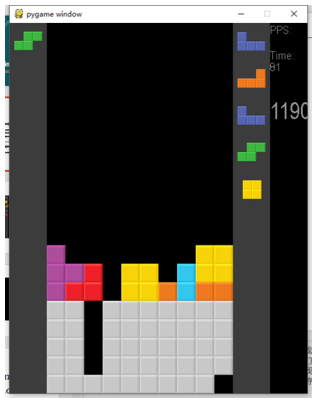

图 2：我们训练的 AI，图中下方的灰色方块就是垃圾快。垃圾块一般一次发送 1、2、4、6、8 行，从画面底端瞬间出现，呈现为中间空出一列的灰色方块。图中的玩家先接受了一组 4 行的垃圾块，后接受了一组 1 行的垃圾块。

但是，当现代规则的俄罗斯方块引入多人游戏模式，并且让玩家能够通过自己的某些特殊操作给对手发送垃圾块后，玩家的目标就成为了在多人对战中存活，或者说，在保证自己不失败的前提下尽量多的对对手进行攻击（即向对手发送垃圾块）。并且，由于现代规则中攻击对手的收益较高，导致很多玩家会将攻击和打击敌人作为自己策略的重点，即使这样会使他们本身变得更容易受攻击而失败。

2.2 操作

经典俄罗斯方块中玩家可以进行以下操作：左移，右移，左转，右转，加速下落；而在现代规则的俄罗斯方块中，玩家除了可以进行经典俄罗斯方块中的操作外，还可以进行硬降、hold 等操作。

- 左移与右移：使当前方块左移或右移一格。
- 左转与右转：使当前方块逆时针或顺时针旋转。
- 加速下落：使当前方块的下落速度增加。
- 硬降（Hard Drop）：使当前方块瞬间降落。
- Hold：如果当前 hold 槽中有方块，则用槽中的方块和当前方块互换；否则，将当前方块存入 hold 槽，将下一个方块变为当前方块。

2.3 SRS

SRS 的全称是 Super Rotation System，是现代版俄罗斯方块中广泛使用的新的旋转系统。它详细地定义了各个方块的出生位置以及旋转方式，尤其是“踢墙”的旋转方式。

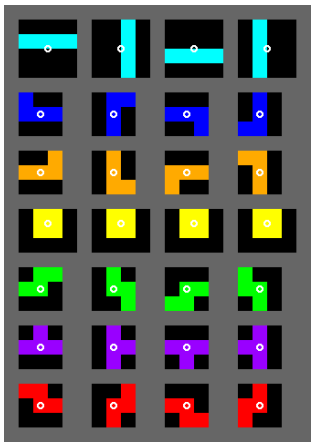

图 3：各个方块的中心点以及旋转方式

2.3.1 关于踢墙（Wall Kick）

踢墙是 SRS 和经典俄罗斯方块中的旋转方式的最重要的区别，而由其衍生出的 T-Spin 等特殊操作更是现代规则中的对战息息相关。踢墙指的是当玩家尝试旋转一个方块，但是方块旋转后的位置和已有的方块重合的情况。在经典规则下，这种旋转被认为是非法的，因此不会产生作用；而在 SRS 规则下，SRS 系统则会对旋转后的方块的周围的位置进行判定，并将

方块移动到可能的位置；如果没有合适的位置，则将与经典俄罗斯方块一样，无法旋转。

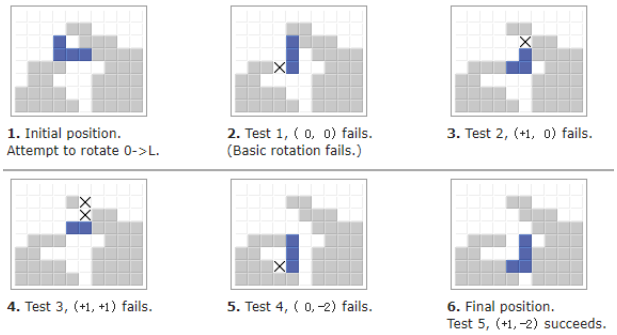

图 4: 在 SRS 规则下，一块“J”块想要在图 1 的情况下进行一次逆时针旋转，但是如图 2 所示，它旋转后的位置和已有的方块重合了。最后，在经过图 3、4、5 的尝试后，它终于成功被放在了图 6 所示的位置中。这一系列的过程在玩家看来都是瞬间完成的，也就是说玩家在图 1 的状态下进行逆时针操作后，方块瞬间变成了图 6 所示的位置。

2.3.2 关于 *-Spin

进行一次 *-Spin，指的是用 * 块（除了 O 块，因为 O 块无法踢墙）在 SRS 规则下进行了一次踢墙操作后成功消行。由于一次 *-Spin 需要同时满足踢墙操作和消行，因此比较难以实现；不过，如果玩家能在对战中完成一次 *-Spin，也将得到相应的收益，比如给对手发送垃圾块等。由于 T 块最容易被用于构建这种 Spin，因此对战中 T-Spin 最为常用，也是玩家最常见的攻击手段之一。

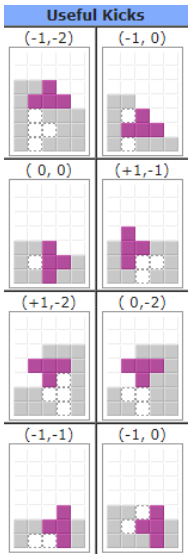

图 5: 图中展示了一些对战中常用的 T-Spin。游戏中，玩家常常会用这些 T-Spin 对对手进行攻击。

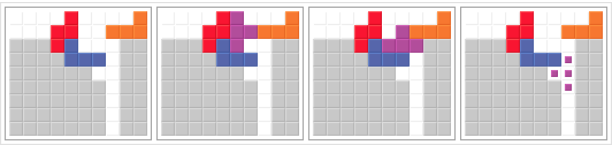

图 6: 图中展示了一次 T-Spin-Tripple 操作——玩家用一次 T-Spin 操作成功地消除了三行。这种一次消除 3 行的 T-Spin 由于需要几个方块的规划和准备以及较强的对局势的预判和观察能力，在对战中较为罕见，也是最强的攻击方式之一，可以一次性给对手发送 6 行垃圾块。

2.3.3 攻击

现代规则给玩家提供了各种攻击对手的方式，比如多消、各种 Spin、连击、全消等。为了简化我们的模型，我们只在游戏中实现了了多消、T-Spin、连击这三种攻击方式。

以下是它们在实际游戏中的攻击强度：

- 多消：除了一次消除一行不会发送垃圾块外，其它消行发送的垃圾块行数为 2^{x-2}。

- T-Spin：只要完成 T-Spin 操作，不管消除多少行，都会发送 $2x$ 行的垃圾块。
- 连击：连击发送的垃圾块行数见表格。连击发送的垃圾块可以和上述两种方式叠加。

表 2: 多消

消除行数	垃圾块行数
1	0
2	1
3	2
4	4

表 3: T-Spin

消除行数	垃圾块行数
1	2
2	4
3	6

表 4: 连击

连击数	垃圾块行数
2	1
3	1
4	2
5	2
6	3
7	3
8	4
9	4
10	4
11	5
12	5

2.3.4 7-Pack

在经典规则下，取得下一个方块是以一种近似随机的方式取得的，这给游戏增加了很大的不确定性，也使永远玩同一局游戏成为不可能。现代规则引入了“7-Pack”，即方块生成将以 7 个方块为一组进行生成，确保每组方块中包含所有种类的方块。

3 问题描述

3.1 规则

AI 了解的游戏规则也就是我们当前设计的规则规范包括：

- SRS 系统下的那 7 个方块在不同情况下旋转后的形状及位置。
- AI 的可操作集合应该与玩家一致（加快下降，向左，向右，左转 90 度，右转 90 度，Hold）。
- 未来几块以及方块产生的算法（7-Pack）。
- 游戏失败条件（可以部分高于顶部但不能某个方块全部都高于顶部）。
- 计分规则。

3.2 目标

AI 的目标自然是在保证生存的情况下以更高的效率给对手发送垃圾块，但是由于我们现阶段难以将当前的 AI 难以直接作用于现有的游戏，因此我们参考了几个现代版俄罗斯方块游戏，设计了特殊的计分公式，将目标定义为处理同等块数时获得更高分数。同时，我们在每次游戏中会统计 AI 进行的特殊攻击操作的数量，我们期待 AI 可以进行更多的特殊操作。

令此次消行数为 n，此次消行前的 Combo 数为 c，此次 T-spin 数为 t（0 或 1），计分公式如下：

$$\text{score} = \text{score} + [2^{n-1} \times 4^t + c] \times 10$$

4 学习算法

基于俄罗斯方块 State Space 极大难以学习的特点，我们最终选择使用 Reinforcement Learning 中的 Ap-

proximate Q-Learning 算法。该算法基于 Q-Learning，在每次 State 更新时，使用当前 State 和 Action 算出一系列的 Features 的值，再利用 Features 组成 Evaluation Function 与 Reward 等，更新该次探索的 State、Next-State、Action 组合的相关 Q-Value。再去更新 Evaluation Funciton 中每个 feature 的权重。核心算法公式如下：

$$\begin{aligned}
\text{transition} &= (s, a, r, s') \\
\text{difference} &= [r + \gamma \max_{a'} Q(s', a')] - Q(s, a) \\
Q(s, a) &= Q(s, a) + \alpha[\text{difference}] \\
\omega_i &= \omega + \alpha[\text{difference}] f_i(s, a)
\end{aligned}$$

其中的 α 为 Learning-Rate, 来自于经验。Evaluation Function 则由一系列的 Feature 值加权而成，将在后续部分涉及。另外，根据探索的特性，我们在学习过程选择新 State 的过程中使用了 Epsilon-Greedy 的算法，该算法会以 的概率随机从所有合法 Action 中选出一个，以 $1 - \epsilon$ 的概率选择最优的 Q-Value 对应的 Action，这样在训练时可以调节其探索-利用的，加快训练过程。
要完成以上算法，我们对游戏进行了建模。

5 算法模型

我们根据现代俄罗斯方块规则的特性建立了方块、State、Action 的模型。

5.1 方块

在俄罗斯方块游戏中，一共有七种方块，模型中包含了七种中的一个字母 (S, Z, J, L, T, O, I) 用于识别方块，由 0、1 构成的二维数组用于记录方块占用的区域，以及二维数组的尺寸。

5.2 State

State 表示了一个某个时刻游戏的状态，是玩家（AI）可以获取到的所有关于游戏环境的信息，包括以二维数组表示的当前的游戏区域、当前的连击数量、分数、正在下落的方块、hold 的方块、未来的方块。

5.3 Action

Action 表示了从一个 State 到另一个 State 的方式，在俄罗斯方块中，玩家可以进行的操作就是将游戏发出的方块放稳在游戏区域的某个位置，基于这个原理，我们设计的 Action 包括方块类型，方块被放置的位置，以及方块的旋转状态。设计了 Action 之后，我们需要从某个 State 找到所有可行的 Action。

5.3.1 获取合法 Action

我们使用 BFS 的算法来寻找所有可行的 Action。对于每一个可能下落的方块，都有向下移动一格、向左右移动一格、逆时针或顺时针选择共计 5 种可能的移动方式。在枚举一系列操作之后，方块最终会到达一个下方有其他块或者是地图边界的位置。如果方块在这一位置不会与游戏区域中的其他已有方块冲突，那么这就是一个可行的 Action。在 BFS 中我们会记录所有的可行 Action 以供之后的选择，并且记录操作的序列。BFS 到达的状态一定是使用了最少操作数，也可以节省时间，符合我们对战的目标。

最初我们的 BFS 从方块下落的出生点开始。这一位置在游戏区域之外，需要大量的特别判断。并且由于位置高，可以进行的操作有很多，整个搜索树相当庞大，也需要大量的时间。随着游戏的进行，游戏区域中的方块可能会变多，整个方块的建筑结构可能会变高。我们观察到建筑结构越高，Action Search 的速度将会越快。之后我们根据这种现象进行了一次优化，将 BFS 的初始位置修改为整个游戏区域有方块的最高行处，让方块直接下落到最高行处再根据我们找到的操作移动。整个搜索树的结点减到了原来的 1/4，搜索速度理论上达到原来的 4 倍。

6 模型特征选择

- landingHeight: 当前块降落地点的高度
- rowTransitions: 每一行中，从无到有、从有到无的变化数
- columnTransitions: 与行变化同理
- holes: 所有洞的数量
- wellDepth: 井的深度
- holeDepth: 所有洞的深度之和
- rowsWithHoles: 有洞的行的数量
- columnHeightP: 每一列的高度
- columnHeightsAvg: 所有列的平均列高
- columnHeightsMax: 所有列中最高的列高
- columnDifference: 相邻列的高度差之和
- rowEliminated: 当前块消除的行数
- tSpinStruct: tspin 结构的数量

6.1 矩阵匹配

矩阵匹配是一个在长 n 格宽 m 格的矩阵中，寻找到几个较小矩阵（长 x 格宽 y 格）出现的次数。直接在矩阵中寻找小矩阵，时间开销为 $O((n-x)(m-y)xy)$。这种算法显然开销太大。如果将矩阵的每一行视为一个字符串，然后再在矩阵中按行使用 KMP 或者 AC 自动机的方法寻找，也只能将每一行匹配的时间降低为 $O((m-y)+y)$，总时间复杂度为 $O((n-x)mx)$。最后我们分析以后采用的算法是对矩阵执行 Hash 操作，并且保证 Hash 整个矩阵使用的时间分别为 $O(nm)$ 和 $O(xy)$。在查找矩阵时，只需要我们对将较大矩阵的各个元素与较小矩阵的 Hash 值进行比较即可。这种方法的时间开销为 $O(nm+xy+(n-x)(m-y))$。

对于我们的问题，即在游戏的 Grid 中寻找特定可以进行 T-Spin 的局部结构，矩阵 Hash 的算法相对于直接寻找理论上提升了近 7 倍的速度。但是由于数据规模实在太小，对比使用自动机的算法的提升可能并不明显。并且本身我们需要寻找的 T-spin 结构就比较多，在寻找 tSpinStruct 时相比于不使用 tSpinStruct 特征的模型出现了明显的延迟。

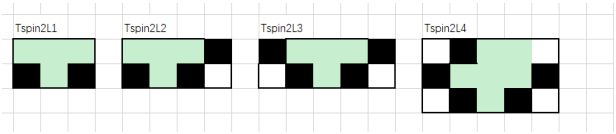

图 7: T-spin 结构图。绿色方块表示必须为空的位置，黑色表示必须有方块的位置，白色表示既可以有方块也可以为空的位置。

6.2 高阶评估函数

传统俄罗斯方块 AI 在分析 feature 的时候可以对每个 feature 进行简单的线性判断，而基于现代规则的俄罗斯方块 AI 则不行。例如，将生存作为第一目标的传统俄罗斯方块 AI 会希望列高度越低越好，因为列高度越低生存的机会越大；而基于现代规则的俄罗斯方块 AI 则可能会希望列高度保持在 4 行左右，因为一次性消除 4 行所能对对方造成的攻击效果值得基于现代规则的俄罗斯方块 AI 冒这个险；除此之外，传统俄罗斯方块的 AI 往往会注重场地的平整，而搭建 T-Spin 则需要破坏这种平整，给 T 块空出一个凹槽。

这些就需要我们的 AI 采用和传统 AI 不同的对 feature 的 evaluation 方式。最终，我们决定使用多项式方式，给各个 feature 加上了他们的平方、三次方和四次方，得到了比之前更好的结果。

7 训练结果评估

在算法实现后，我们用 1 阶评估函数和 4 阶评估函数分别从零开始训练了 1000 个方块，同时记录每种函数在第一局游戏后，第 50 个方块后，第 200 个方块后，第 1000 个方块后的权重。为了评估这些训练结果，我们仿照真实对战环境建立了垃圾块模拟的功能。在 AI 游戏时，程序会在每个新方块生成时以不同概率随机生成不同行数的垃圾块，将场地内现有的方块往上顶。我们在垃圾块环境下对每个训练结果测试了

五次（暂时关闭学习算法的更新），对相关统计数据取平均值进行分析以对 AI 的能力获得一个较全面的认识。每一局的统计数据包括死亡前消除的行数、得分的效率 (总得分/处理的方块数)、多层消除的比率（2 4 消）、连击的数量以及最终得分等。

表 5: 使用一阶评估函数在训练了不同次数后的五次垃圾行环境测试中的平均指标

	得分	方块	消行	单消率	二消率	三消率	四消率	二连率	三连率	四连率	五以上
1 局	434	91	30.2	80.80%	17.60%	1.60%	0.00%	12.80%	8.80%	0.00%	0.80%
50 次	554	103.8	36.8	96.63%	3.37%	0.00%	0.00%	12.92%	4.49%	2.25%	0.56%
200 次	1392	217.2	92.4	79.78%	15.57%	3.28%	1.37%	15.57%	4.92%	0.00%	1.09%
1000 次	3008	440.6	207.8	79.08%	16.30%	3.77%	0.85%	14.60%	5.96%	1.22%	0.49%

表 6: 使用四阶评估函数在训练了不同次数后的五次垃圾行环境测试中的平均指标

	得分	方块	消行	单消率	二消率	三消率	四消率	二连率	三连率	四连率	五以上
1 局	888	140.6	55.8	92.25%	7.36%	0.39%	0.00%	11.63%	6.20%	1.94%	1.16%
50 块	1122	165	70	73.66%	20.99%	3.44%	1.91%	6.11%	4.20%	2.29%	1.91%
200 块	1848	255.4	117.6	67.88%	24.09%	5.11%	2.92%	11.68%	3.41%	0.73%	1.70%
1000 块	3704	509.8	245.6	66.27%	24.97%	5.92%	2.84%	13.96%	6.15%	1.66%	0.71%

8 总结

由于我们选择了一个基于 pygame 包开发的俄罗斯方块游戏，我们很大程度上依赖于这个游戏的初始设计。并且为了模拟对战，我们人为地修改了规则，特别是一些特殊操作的得分以鼓励 AI 主动去进行这些操作。但是实际上这些操作的得分在各个游戏中是不一样的。可能我们的模型并不能在其他俄罗斯方块的游戏中击败多数人类玩家。

另一方面，受限于 Approximate Q-learnning 的方法，我们只能寻找当前下落方块和已经 hold 的方块的 leagal action。AI 在游戏时可能会主动搭建诸如 T-spin 等特殊操作所需要的结构，但是并不能在对应的方块到来时使用。也就是说，我们的 AI 并不懂得在游戏中“运营”。

在 Action Search 和矩阵匹配阶段，我们的算法设计相对简单，并不高效。这是因为我们使用了 python 进行开发，并不能很好地利用一些底层的优化来加速运算。

在训练中，我们的游戏和 AI 是同一线程，只有在完成 Action Search 并且找到最佳 Action 之后，AI 才会移动方块到确定的地点。在真实的对战中，方块是自动下落的，并不会等待 AI 决定最佳 Action。

如果可能，我们会将 AI 从 pygame 中解放出来，使用更加高效的工具重构为 Double Q-learning 或者 Deep Q-learning 的方法学习游戏的 AI，并尝试在线上对战平台与真实人类玩家进行对战。**?**

参考文献

[1] Christophe Thiery, Bruno Scherrer, Christophe Thiery, Bruno Scherrer. Improvements on Learning Tetris with Cross Entropy. International Computer Games Association Journal, ICGA, 2009, 32. ffinria-00418930f

[2] Matt Stevens, Sabeek Pradhan. Playing Tetris with Deep Reinforcement Learning

[3] Ryan Heise. Playing TETRIS using the PageRank Algorithm. https://www.ryanheise.com/tetris/tetris_artificial_intelligence.html

[4] Misakamm, 传统规则俄罗斯方块 AI 技术介绍. http://misakamm.com/blog/482