The 33rd Annual Conference of the Japanese Society for Artificial Intelligence, 2019

Teaching Reinforcement Learning and Computer Games with 2048-Like Games

Hung Guei* Ting-Han Wei* I-Chen Wu*

* Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan

2048-like games are a family of single-player stochastic puzzle games, which consist of sliding numbered-tiles that combine to form tiles with larger numbers. Notable examples of games in this family include Threes!, 2048, and 2584. 2048-like games are highly suitable for educational purposes due to their simplicity and popularity. Numerous machine learning methods have been proposed for 2048, which provide a good opportunity for students to gain first-hand experience in applying these techniques. This paper summarizes the experience of using different 2048-like games, namely Threes! and 2584, as pedagogical tools for teaching reinforcement learning and computer game algorithms. With two classes of graduate level students, the average win rates for 2584 and Threes! reached 96.1% and 93.5%, respectively. The course designs were also well received by students, with 4.21/5 and 4.35/5 points from student feedbacks.

1. Introduction

2048 is a single-player stochastic puzzle game introduced as a variant of 1024 and Threes! [Cirulli 2014]. 2048 is easy to learn and to play reasonably well, yet mastering the game is far from trivial. Many machine learning methods have been applied to 2048 such as the well-known Temporal Difference Learning [Szubert 2014]. As a teaching tool, 2048's popularity can increase student engagement, while the existing machine learning methods for it provide a well-established basis to educate from.

We have summarized the experience of using 2584, a variant of 2048, as the pedagogical tool for teaching reinforcement learning and computer game algorithms [Guei 2018]. In this paper, we extend our previous work, propose a new course design for Threes! and summarize the experiences and list the differences between using Threes! and 2584 in the curriculum. With two consecutive classes of graduate level students, the average win rate for 2584 and Threes! student projects reached 96.1% and 93.5%. The course projects were also well received by students, with 4.21/5 and 4.35/5 points from feedbacks.

This paper is organized as follows. Section 2 introduces 2048-like games and reviews existing related techniques. Section 3 shows how teaching material is designed with Threes! in our courses, summarizes student results, and compares differences with previous 2584 lectures. Section 4 provides student feedback and makes concluding remarks.

2. Background and Related Techniques

In this section, we will first introduce some 2048-like games, then briefly review important relevant algorithms and techniques.

2.1 Single-player 2048-like Games

We first describe the game, 2048[1], and then 2584[2] and Threes![3], the first two of which are derived from Threes!. 2048 is a 4×4 puzzle (16 grids), starting with two tiles. Each grid on the puzzle is either empty or contains a numbered tile with a value that is a power of two. The objective is to slide the puzzle such that the tiles merge into larger tiles.

Upon the player sliding the puzzle, all tiles will slide to the specified direction as far as possible. Adjacent tiles with the same value, say v-tile, will be merged into a larger tile, $2v$-tile, and the player will receive a reward $2v$. A sliding direction is illegal if the puzzle remains unchanged after sliding.

The environment immediately generates a new tile after the player slides the puzzle. The new tile can be either a 2-tile or a 4-tile, with the probabilities of 0.9 and 0.1, respectively. The new tile will be randomly placed at any empty grid.

After the new tile has been added, the player continues to slide the puzzle. This process repeats until no legal direction is possible. The final score is the sum of rewards gained from merging. Players win if a 2048-tile is generated.

For 2584, tiles are labeled by the numbers in the Fibonacci series, instead of powers of two. The environment generates 1-tiles and 2-tiles with the probabilities of 0.9 and 0.1. However, instead of merging tiles with the same value, two adjacent tiles whose values are adjacent numbers in the series are merged. Players win if a 2584-tile is generated.

Threes! is a game originally developed by Vollmer and Wohlwend. In Threes!, the game starts with nine initial tiles. The sliding distance is at most one. A 1-tile and a 2-tile can be merged into a 3-tile; for tiles with values v between $3 \leq v < 6144$, two v-tiles can be merged into a $2v$-tile. The 6144-tile is designed so that it can no longer be merged.

In Threes!, a *hint* is observable for the next generated tile before sliding the puzzle. The generated tile will be randomly placed at a newly cleared empty space (generated by sliding a 1-tile) on the opposite side of the last sliding direction. The type of generated tiles is controlled by the so-called *bag rule* and *bonus rule*. Consider a bag of 12 tiles composed of equal amounts of 1-, 2-, and 3-tiles. A tile is randomly selected and removed from the bag during tile generation, until the bag is empty and then refilled. The bonus rule states that when the largest tile on the current puzzle,

Contact: I-Chen Wu, Department of Computer Science, National Chiao Tung University, Taiwan, +886-3-5731855, +886-3-5733777, icwu@cs.nctu.edu.tw

[1] Available at https://gabrielecirulli.github.io/2048/

[2] Available at https://davidagross.github.io/2048/

[3] Available at http://asherv.com/threes/ and http://threesjs.com/

the $v_{\max}$-tile, is at least a 48-tile, there is a probability of $1/21$ to generate a bonus v_+-tile where $6 \leq v_+ \leq 1/8 \times v_{\max}$ and each possible value for v_+ has equal probability. For the remaining $20/21$ cases, bag tiles are generated.

The game ends when the player has no legal direction to slide. The final score is the sum of $3^{\log_2(v/3)+1}$ of all v-tiles with $v \geq 3$. In this paper, players are said to win if a 384-tile is generated.

2.2 Two-player 2048-like Games

In this paper, 2048-like games are modified into a two-player games as follows. While one player is still called *player*, his opponent is called *adversary* to play the role of an antagonistic environment that makes the player more difficult to play, i.e., the player maximizes the score, while the adversary minimizes it.

Thus, the modified two-player game begins with the adversarial side. First, the adversary places some tiles on an empty puzzle. Then, the player and the adversary take turns sliding the puzzle or placing a tile. The game ends when the player is unable to slide.

2.3 Generic Framework of 2048-like Games

All the puzzles in 2048-like games can be categorized into two kinds of states: *before-states* and *after-states*. An instance of a 2048-like game begins with a special before-state called the *initial state*. The player performs an *action*, i.e., sliding the puzzle, to the before-state, upon which the before-state will transform into an after-state. The environment then makes changes to the after-state, which renders it into another before-state for the next *time step*. The game continues until reaching a *terminal state*, which is a before-state for which the player is unable to perform any actions.

2.4 Techniques Related to 2048-like Games

(1) Tree Search

The original single-player 2048 is an expectimax game. In the case of a two-player game where the adversary can determine both the type and the position of new tiles, the game follows the minimax paradigm. Conversely, if the type of new tile is decided randomly and the adversary can only determine the position, the game conforms to the expectiminimax paradigm. In practice, heuristic or value functions are combined with tree search since there tends to be insufficient time to expand to leaf nodes.

(2) Temporal Difference Learning

Temporal Difference Learning (TD) is a reinforcement learning method [Sutton 1998] which was first applied to 2048 in 2014. [Szubert 2014]. The simplest TD(0) updates the value function $\mathrm{V}(s_t)$ with the prediction error $\delta = r_t + \gamma \mathrm{V}(s_{t+1}) - \mathrm{V}(s_t)$ from subsequent values through $\mathrm{V}(s_t) \leftarrow \mathrm{V}(s_t) + \alpha\delta$, where t is the time step, α is the learning rate, and γ is the discount factor. The value function $\mathrm{V}(s_t)$ can be viewed as the expected return of a given state s_t. Therefore, a policy $\pi(s_t)$ can be derived as $\pi(s_t) = \underset{a_t}{\operatorname{argmax}}\big(r_t + \mathrm{P}(s'_t, s_{t+1})\mathrm{V}(s_{t+1})\big)$, where $\mathrm{P}(s'_t, s_{t+1})$ is the probability of transition from an after-state s'_t to a state s_{t+1}.

The general form, TD(λ), updates the value function with all subsequent errors with a trace decay parameter λ. Higher λ values increase the proportion of prediction error from more distant states and actions [Sutton 1998]. TD(0.5) has been successfully applied to 2048 [Yeh 2017] [Jaśkowski 2017].

Forward update and backward update are both possible for TD implementation. Forward update here refers to the scheme where all states are updated in order of an episode from initial state to terminal state; backward update reverses the order. Backward update is slightly better than forward update when the training episodes are the same [Matsuzaki 2017a]. Also, backward update is easier for students to understand and implement [Guei 2018].

Multi-stage Temporal Difference Learning (MS-TD) divides the entire episode into several stages, each with a unique function approximator. It has been applied to 2048 and has successfully reached the first-ever game with 65536-tile by computer program [Wu 2014] [Yeh 2017]. Several other implementations have also been investigated, such as finding more optimal ways to divide stages [Jaśkowski 2017] [Matsuzaki 2017b].

Temporal Coherence Learning (TC) is a variant of TD with adaptive learning rates, which has been applied to 2048 [Jaśkowski 2017]. The update amount is controlled by the parameter α and the coherence $\sum|\delta|/|\sum\delta|$, where δ is the TD error of each update. Therefore, the update amount decreases if the TD error δ starts to oscillate between positive and negative values.

(3) Function Approximator

The state spaces of 2048-like games are quite large, which makes it unaffordable for current computers to store the entire state space with a direct mapping table. Therefore, a function approximator can be an efficient way to obtain state values.

N-tuple networks are a well-known function approximator for 2048-like games since 2014 [Szubert 2014]. An n-tuple network estimates the value of a state by extracting features from the state and accumulating feature values, where features are usually sub-puzzles in 2048-like games. The performance of n-tuple networks is highly correlated with feature design. Good configurations are not trivial, and have been investigated as a topic of research [Yeh 2017] [Matsuzaki 2016]. Each feature maps to an entry of a lookup table. The value function is therefore $\mathrm{V}(s) = \sum_i^m \mathrm{LUT}[\phi_i(s)]$, where LUT is the weight table, m is the total number of features, and ϕ_i refers to the i^{th} feature. Features are updated as follows, $\mathrm{LUT}[\phi_i(s)] \leftarrow \mathrm{LUT}[\phi_i(s)] + (1/m)\alpha\delta$.

Deep neural networks (DNN) and convolutional neural networks (CNN) can also be used as function approximators for 2048 [Guei 2016] [Wei 2019]. Compared with n-tuple networks, DNNs usually consist of fewer weights overall, but more weights are involved for each output. At the time of writing, strong 2048-like game programs currently use n-tuple networks as the function approximator [Yeh 2017] [Jaśkowski 2017]. Using DNNs efficiently for 2048-like games is still an open research topic.

3. Course Design and Student Results

Theory of Computer Games is a course taught by Professor I-Chen Wu at National Chiao Tung University, Hsinchu, Taiwan. The course is designed for graduate level students, and the prerequisites are Algorithms and Data Structures. Also, students are expected to be moderately proficient at programming.

Students are required to develop a game-playing program as the term project during the semester, which spans about five months. The overall program is broken down into six projects, which are built on top of each other. In the series of projects, students are required to develop their program step by step.

In the subsequent subsections, we will first give a short summary of each project, and then compare the difference of using 2584 and Threes! in the curricula for 2017 and 2018, respectively.

- Project 1: Learning to use the framework
- Project 2: Train the player using TD and n-tuple networks
- Project 3: Solve a reduced game by expectimax
- Project 4: Further improve the performance of the player
- Project 5: Design the adversarial environment
- Project 6: Participate in the final tournament

3.1 Learning to Use the Framework

First of all, students need to set up the framework and prepare the training environment for the coming projects in two weeks. We provide a demo of 2048 as the framework[4], where the students are expected to modify the rules to the target game. Since the original rules for Threes! is complex, we simplify them as in Table 1.

Table 1. Rule simplifications in Project 1

2584	The same as original 2584.
Threes!	Bag size is set to 3; no bonus tiles. New tiles can be placed at any empty position on the opposite side of the last sliding direction.

3.2 Train the Player Using TD and N-tuple Networks

Students are required to train a strong player using TD with n-tuple networks in one month. The main purpose is to ensure that students understand the mechanism of TD and n-tuple networks.

The environments follow that described in Table 1. We recommend students to start with the simplest feature design, which contains 4 rows and 4 columns, for a total of 8 standalone 4-tuple patterns. The initial learning rate α is 0.1.

Table 2. Average win rate and n-tuple in Project 2

	Avg. win rate	Avg. n for n-tuples
2584 34 students	$96.1\% \pm 4.2\%$ 10 reached 100%	4.6 ± 0.8 # of {4,5,6}-tuple: {20,7,7}
Threes! 43 students	$93.5\% \pm 11.2\%$ 5 reached 100%	5.6 ± 0.8 # of {4,5,6}-tuple: {9,1,33}

We use the win rate to analyze the performance of student programs. As shown in Table 2, reaching the 2584-tile in 2584 seems to be easier than reaching the 384-tile in the simplified version of Threes!.

The average n of n-tuples is also different. Most of the students applied a 6-tuple configuration in Threes! since it has better performance. However, not so many students tried 6-tuples in 2584 since the tile indices can easily grow to more than 20, resulting in a much higher memory cost.

3.3 Solve a Reduced Game by Expectimax

The aim of this project is to familiarize the students with the expectimax paradigm. Students need to use the expectimax search to solve a reduced game with a puzzle size of 2×3. The time allocated for this project is at most one month.

The environments are still the same. However, there is only one initial tile for Threes!. In project 3, student solvers are required to answer the expected value of given input questions. The average numbers of solved questions are listed in Table 3.

Solving 2×3 Threes! seems to be harder than 2×3 2584, as shown in Table 3. There are two potential reasons for this. First, we observed that the students ran into difficulties implementing expectimax search with hints, which was unique to Threes!. Since the framework we provided was designed for 2048, students needed to program hint processing on their own. Second, we introduced some changes to Threes! in 2018 to increase the project difficulty, so that we can better discriminate student performance.

Table 3. Average questions solved in Project 3

	Avg. solved
2584 34 students	$99.9\% \pm 0.5\%$ 31 reached 100%
Threes! 35 students	$92.5\% \pm 15.5\%$ 24 reached 100%

In 2018, students need to answer not only the expected value, but also the best and the worst value. The best value occurs when the player is extremely lucky, i.e., the environment coincidentally generates tiles which leads the player to the highest score under oracle play. The worst value is exactly the opposite. Although this definition is not difficult to understand, more specifications tend to lead to more avenues of error for students.

3.4 Further Improve the Performance of the Player

The objective is encouraging students to improve their player with optional methods. Project 4 is similar to Project 2, but with more stringent environments, which we show in Table 4.

Table 4. Rule modifications in Project 4

2584	*1-tiles and 3-tiles* are generated with probabilities of *0.75 and 0.25*, respectively.
Threes!	*Bag size and bonus tiles follow original*. New tiles can be placed at any empty position on the opposite side of the last sliding direction.

Note: Differences between Project 4 and Project 2 are in italic.

Students are required to retrain their players under the new environments. Several possible improvements are needed to achieve a comparable level of performance. Regardless of the method chosen, Project 4 is expected to be finished in one month.

Our first suggestion to students is adding the expectimax search that has already been implemented in Project 3. The second is using complex network structures, isomorphism of features may also be needed to avoid high memory usage and to speed up training. The third is decreasing the learning rate if they have not yet done so. The last suggestion is trying some advanced TD methods, such as using TD(λ), TC, or MS-TD.

Table 5. Average win rate, n-tuple, and depth in Project 4

	Avg. win rate	Avg. n for n-tuples	Avg. depth*
2584 31 students	$82.0\% \pm 11.3\%$ 12 reached 90%	5.5 ± 0.7 # of {4,5,6}-tuple: {4,8,19}	2.6 ± 0.8 25 applied
Threes! 40 students	$88.8\% \pm 15.1\%$ 26 reached 90%	6.3 ± 0.7 # of {4,5,6,7†}-tuple: {2,0,21,17}	1.5 ± 0.9 10 applied

* The depth for no extra search is 1; for an additional 1-ply search is 3.
† Extra encoding such as hints is counted as 1.

The final results of Project 4 are shown in Table 5. Contrary to previous results, students got better performances in Threes!. One reason is that the rule changes for 2584 is much more difficult, as 3-tiles are unmergeable with neither 1-tiles nor other 3-tiles, and are generated with a high probability of 0.25. Another possible reason for this result is the usage of large networks. Many students encoded the hint for the next generated tile into their network,

[4] https://github.com/moporgic/2048-Framework/branches/

which greatly improved the performance. Since the tile is limited to 6144-tile (14^{th}), students can also enlarge the network easily.

Since the expectimax project for Threes! in 2018 was quite difficult, relatively fewer students added expectimax into their player. The average win rate might have been higher if expectimax was applied more often.

3.5 Design the Adversarial Environment

The objective of Project 5 is to build an adversary, which is a clear departure from previous projects where students work solely from the perspective of the player. The adversary in two-player Threes! can control both the type and position of a new tile, which conforms to the minimax paradigm. In contrast, the adversary in 2584 can only control the position, while the type is randomly generated, which conforms to an expectiminimax paradigm.

One month is given for Project 5. The best practice is retraining networks for player and adversary under the new paradigms. Note that it is possible to make an adversary by reusing the network and choosing the minimum value. However, the performance may drop slightly due to paradigm mismatch.

Table 6. Average win rate, n-tuple, and depth in Project 5

	Avg. win rate	Avg. n for n-tuples	Avg. depth*
2584 31 students	87.5% ± 14.7% 18 reached 90%	5.5 ± 0.7 # of {4,5,6}-tuple: {4,7,20}	2.9 ± 0.5 29 applied
Threes! 37 students	52.7% ± 35.6% 9 reached 90%	6.4 ± 0.7 # of {4,5,6,7†}-tuple: {2,0,17,18}	2.4 ± 0.9 25 applied

* The additional 1-layer search of reusing the network is not included.
† Extra encoding of the network such as hints is counted as 1.

We also grade the adversary by its win rate, defined as the rate at which the player loses. The results are listed in Table 6. The variance of adversary performance in Threes! is quite large; this was unexpected since the minimax setting should have been easier to handle than the expectiminimax setting for 2584. It is possible that hint processing in search is much more complex, mistakes were made during network reuse, or minimax search was implemented incorrectly.

3.6 Participate in the Final Tournament

All students are required to participate in the final tournament. To determine the result between two students, two games are necessary: each student acts as the player and the adversary exactly once. The student whose player gets the higher score wins and receives one point. The ranking is then determined by the total number of points.

30 students participated in the 2584 tournament in 2017, and 40 students in the Threes! tournament in 2018. The top ranked student got 28 points in 29 matches in 2017; the winner for 2018 got 109 points in 117 matches for Threes!. It is worth mentioning that the top ranked programs adopted different strategies. The 1^{st} place 2584 program applied a simple 4-tuple network, while the 1^{st} place Threes! program applied a complex 7-tuple network.

Another difference between the above two tournaments was the wide use of advanced TD methods in 2018. Only a few students applied TC in 2017. However, many advanced TD methods such as TC, TC(λ), MS-TD appeared in 2018. This increased the strength of the highest-performing programs and intensified the competition between students.

4. Summary

For the popularity and simplicity of 2048, 2048-like games have become a staple application for reinforcement learning in the term project starting from 2014. From positive student feedbacks (4.21/5 and 4.35/5 points), experience sharing of students[5], and gradual improvement in program strength over years of using 2048-like games as term projects, we think 2048-like games are highly suitable for teaching computer game programming techniques and machine learning, especially for reinforcement learning. It can be a good pedagogical tool to motivate young minds in joining our field and community.

References

[Cirulli 2014] G. Cirulli, “2048, success and me”, Retrieved from http://gabrielecirulli.com/articles/2048-success-and-me, 2014.

[Guei 2016] H. Guei, T.-H. Wei, J.-B. Huang, and I-C. Wu, “An Empirical Study on Applying Deep Reinforcement Learning to the Game 2048”, Workshop Neural Networks in Games in the International Conference on Computers and Games, Springer, 2016.

[Guei 2018] H. Guei, T.-H. Wei, and I-C. Wu, “Using 2048-like Games as a Pedagogical Tool for Reinforcement Learning”, International Conference on Computers and Games, Springer, 2018.

[Jaśkowski 2017] W. Jaśkowski, “Mastering 2048 with delayed temporal coherence learning, multi-stage weight promotion, redundant encoding and carousel shaping”, Transactions on Computational Intelligence and AI in Games, IEEE, 2017.

[Matsuzaki 2016] K. Matsuzaki, “Systematic selection of N-tuple networks with consideration of interinfluence for game 2048”, Technologies and Applications of Artificial Intelligence, IEEE, 2016.

[Matsuzaki 2017a] K. Matsuzaki, “Developing a 2048 Player with Backward Temporal Coherence Learning and Restart”, Advances in Computer Games, Springer, 2017.

[Matsuzaki 2017b] K. Matsuzaki, “Evaluation of Multi-staging and Weight Promotion for Game 2048”, 高知工科大學紀要テクニカルレポート, 2017.

[Sutton 1998] R. S. Sutton, and A. G. Barto, “Reinforcement learning: An introduction”, MIT press, 1998.

[Szubert 2014] M. G. Szubert, and W. Jaśkowski, “Temporal difference learning of N-tuple networks for the game 2048”, Computational Intelligence and Games, IEEE, 2014.

[Wei 2019] T.-J. Wei, “A Deep Learning AI for 2048”, Retrieved from https://github.com/tjwei/2048-NN, 2019.

[Wu 2014] I-C. Wu, K.-H. Yeh, C.-C. Liang, C.-C. Chang, and H. Chiang, “Multi-stage Temporal Difference Learning for 2048”, Conference on Technologies and Applications of Artificial Intelligence, Springer, 2014.

[Yeh 2017] K.-H. Yeh, I-C. Wu, C.-H. Hsueh, C.-C. Chang, C.-C. Liang, and H. Chiang, “Multistage Temporal Difference Learning for 2048-Like Games”, Transactions on Computational Intelligence and AI in Games, IEEE, 2017.

[5] Some experiences of students are shared (in Chinese) at http://blog.sharknevercries.tw/2018/01/23/2584-AI and https://junmo1215.github.io/tags.html#2584-fibonacci-ref.