A Tetris AI with an Adventurous Strategy Based on Attribute Theory Method

XU Pu
College of Information Engineering
Shanghai Maritime University
Shanghai 201306, P. R. China
Email: xupuxps@gmail.com

FENG Jiali
College of Information Engineering
Shanghai Maritime University
Shanghai 201306, P. R. China
Email: jlfeng@shmtu.edu.cn

***Abstract*—During recent years, an amount of researchers are becoming interested in the popular video game Tetris. However, most of the researchers concentrate on eliminating as many rows as possible only, without considerations on how many rows are eliminated within each action. In this paper, we simulate human players' behavior and try to increase the average rows eliminated each action as far as we can, and during this procedure we basically use Attribute Theory Method. We call it adventurous strategy because it is venturesome in some respect. We also introduce a danger evaluation to counteract the adverse factors of this strategy.**

***Keywords*—Tetris; Artificial Intelligence; Adventurous Strategy; Attribute Theory Method.**

I. INTRODUCTION

Tetris is a video game which was originally invented by Russian programmer Alexey Pajitnov at about 1985. It is still very popular by now. [1][2]

The game area of Tetris, usually called *board*, is a matrix of cells with a height of M and width of N (original 20 by 10). The blocks, usually called *Tetrominos*, fall down from the top of the board one after another. They keep falling down step by step until they touch the bottom of the board or other Tetrominos which have already been placed. There are seven kinds of Tetrominos in the original Tetris, which are shown in Fig. 1. From left to right are: I, J, L, O, S, T and Z. Each of the Tetrominos occupies four cells. The player can move Tetrominos horizontally or rotate them by 90 degrees in order to put it to a proper place. After some Tetromino is placed, if all the cells in row are occupied, the whole row would be removed, usually called *eliminated*. Also, all the blocks above slip down by one line. For more details, Fahey's website [2] would be a good reference.

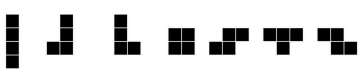

Fig. 1. Seven kinds of Tetrominos: I, J, L, O, S, T and Z

There are two distinct sets of rules in Tetris, one piece and two pieces, respectively equal to knowing the current Tetromino only and knowing the current and the next both. Obviously, two piece algorithms consider more hence are more complex and effective.

Researchers have been working on Tetris during the last two decades, but most works are done since 2000. Not only multiple classic methods have been applied to solve Tetris, but also some academic works done. Tetris can be considered as a sequential strategic decision problem and thus matches Markov Chain Model.

By now the best algorithm can eliminate over 650, 000 rows on average [2]. Top algorithms, such as ones by Pierre Dellacherie [2], Colin Fahey [2], Roger Llima [3], Niko Böhm [1], Yang Gao [4] and Xingguo Chen [5] are claimed as with capacity of eliminating tens of thousands of lines. Papers by Amine Boumaza [6] and Christophe Thiery [7] are good references for comparing all the popular algorithms.

II. TECHNIQUE REVIEW

A. Previous methods

Before discussing our adventurous strategy, we would review some major methods that have been used in Tetris, on which our strategy based.

1) Features and evaluation function: Almost all the best algorithms are using features and evaluation function, including those from Dellacherie, Fahey and Böhm. Actually we consider them to be the same because they are all based on the global evaluation of the Tetris board, including choosing features and assembling the weighted features by a linear function. The only difference is Dellacherie and Fahey chose features by hand, while Böhm and the others chose by some kind of machine learning technique.

Some popular features, such as peak_height, landing_height, number_of_holes, row_transition, column_transition and eroded_piece_cells (the product of rows that the current Tetromino can eliminate and the cells of the current Tetromino can be eliminated in this action) are widely used by several researchers. Christophe Thiery summarized almost all the features used by several researchers in his paper [7]. We will use some of them instead all as the others are actually not that important. Samuel Sarjant [8] explained the reason for not using them. This can decrease the complexity of the algorithm and the evolutionary progress. Dellacherie chose only 4 features but can still beat Böhm, who chose over 12 features. Thus, choosing good features are extremely

important, but we will make use of all these previous works to simplify ourselves'.

2) Semi guided sub state: Unlike global evaluation, Samuel Sarjant introduced a neighborhood evaluation concept named *sub state* in his paper [8]. Its basic idea is to find some sub area which fit the current Tetromino best.

B. Attribute Theory Method (ATM)

Attribute Theroy Method (ATM) is an artificial intelligence (AI) method which was invented by Prof. Feng Jiali from Shanghai Maritime University. Here we simply review some important facts in ATM [9].

1) The Qualitative Mapping Model of the Transformation from Quantitative Attribute to Qualitative Attribute: Give a definition first: note the qualitative criterion of attribute $p_i(x)$ to be $c_i(x) = (\alpha,\beta)_i \in D_d$, if and only if, for any $d_a \in c_i(x)$, its corresponding $q_d(x)$ can be qualified as $p_i(x)$, which means, $c_i(x)$ being the qualitative criterion of attribute $p_i(x) \iff q_d(x) = p_i(x) \leftrightarrow d_a x \subseteq c_i x$.

Then, the relationship between the quatitative attribute and qualitative attribute can be defined as followed: let a(u) be an attribute of some object, $x \in X \subseteq R$ be a quantity of a(u), and $p_i(u) \in P_u$ be a quality of a(u), $(\alpha_i,\beta_i) \in \Gamma$ be the qualitative criterion of the quality $p_i(u)$. Then we consider the mapping $\tau : X \times \Gamma \to 0,1 \times P_u$ to be the qualitative mapping whose qualitative criterion is (α_i,β_i), if for any $x \in X$ and $(\alpha_i,\beta_i) \in \Gamma$, there exists a quality $p_i(o) \in P_u$, which validate the following equation and whose qualitative criterion is (α_i,β_i):

$$\tau(x,(\alpha_i,\beta_i)) = \tau(x \underset{?}{\in} (\alpha_i,\beta_i)) =$$

$$\begin{cases} p(u) & \iff x \in (\alpha_i,\beta_i) \\ \sim p(u) & \iff x \notin (\alpha_i,\beta_i) \end{cases} = \mu_i(x) p_i(u)$$

in which, $\mu_i(x) = \begin{cases} 1 & \iff x \in (\alpha_i,\beta_i) \\ 0 & \iff x \notin (\alpha_i,\beta_i) \end{cases}$ is the boolean value of quality $p_i(u)$ corresponding to the quantity x. $x \underset{?}{\in} (\alpha_i,\beta_i)$ is the question *"is x belong to the interval* $[\alpha,\beta]$*?"*, and $\tau(x \underset{?}{\in} (\alpha_i,\beta_i))$ is the boolean solving operator of that question.

2) The Attribute Computing Network: Let $a(u) = \bigwedge_{i=1}^{n} a_i(u)$ be the conjunctive attribute of all the attributes of object u, which noted as $a_i(u)$ $(i \in (1,\cdots,n))$, $x = (x_1,\cdots,x_n \in X = X_1 \times \cdots \times X_n \subset R^n$ be the quantity attribute of a(u), $p_{ij}(u)$ be the quality of the sub attribute $a_i(u)$, and $[\alpha_{ij},\beta_{ij}] \subset X_i$ be the qualitative criterion of $p_{ij}(u)$. Then we note $S_{p_v} = [\alpha_v,\beta_v) = [\alpha_{1j_1},\beta_{1j_1}) \times \cdots \times [\alpha_{nj_n},\beta_{nj_n})$ as a n-demension qualitative criterion, which can be considered as a hyperparallel and formed by n $[\alpha_{ij_i},\beta_{ij_i}) \in \{[\alpha_{ij},\beta_{ij})\}$, which are respectively from the ualitative criterion set $\{[\alpha_{ij},\beta_{ij})\}$, whose index is i. We also note $(ij_1,\cdots,ij_n)$ as a assemble of them, $v = v(ij_1,ij_n) \in 1,\cdots,m^n$ as the serial numbers of this assemble, and $G(\alpha_v,\beta_v$ as the grid formed by all the orthometric n-demensional hyperparallels $[\alpha_v,\beta_v]$, whose total number is m^n. Also, let $p_v(u) = \bigwedge_{ij_1,\cdots,ij_n} p_{ij_i}(u)$ be an integrated attribute whose qualitative criterion is $[\alpha_v,\beta_v]$, and $\Gamma = [\alpha_{v]},\beta_v]$ be the cluster of the qualitative criterion. We call the mapping $\tau : X \times \Gamma \to 0,1$ be the qualitative mapping whose qualitative criterion is a grid $G([\alpha_v,\beta_v])$, if for any $x \in X$, there exists $[\alpha_v,\beta_v] \in \Gamma$ and a quality $p_v \in P$ whose qualitative criterion is $[\alpha_v,\beta_v]$, can validate the following equation:

$$\Gamma\left(\begin{matrix} & [\alpha_{11},\beta_{11}] & \cdots & [\alpha_{1m},\beta_{1m}] \\ (x_1,\cdots,x_n), & \vdots & [\alpha_{i_kj_l},\beta_{i_kj_l} & \vdots \\ & [\alpha_{n1},\beta_{n1}] & \cdots & [\alpha_{nm},\beta_{nm}] \end{matrix}\right)$$

$$= \bigvee_{j_l=1}^{m} \bigwedge_{i_k=1}^{n} \tau\left\{ (x_1,\cdots,x_n) \underset{?}{\in} \begin{matrix} [\alpha_{i_1j_1},\beta_{i_1j_1}] \times \cdots \\ \times [\alpha_{i_kj_l},\beta_{i_kj_l}] \times \cdots \\ \times [\alpha_{i_nj_m},\beta_{i_nj_m}] \end{matrix} \right\}$$

$$= \bigvee_{j_l=1}^{m} \left\{ \cdots \left\{ \bigwedge_{i_k=1}^{n} \tau_{v(i_1j_1,\cdots,i_kj_l,\cdots,i_nj_m)}(x) \right\} \right\}$$

in which, $\tau_{v(i_1j_1,\cdots,i_kj_l,\cdots,i_nj_m)}(x) = \begin{cases} 1 & \iff x \in [\alpha_v,\beta_v] \\ 0 & \iff x \notin [\alpha_v,\beta_v] \end{cases}$. Then we call the two equations above be the qualitative mapping judging whether an object u with a vector x have a quality $p_v(u)$ or not, and the qualitative mapping whose qualitative criterion is $[\alpha_v,\beta_v]$.

III. An adventurous strategy

Although some algorithms can eliminate hundreds of thousands of rows within a game, if you look at the procedure of the previous methods mentioned above, especially Fahey's Standard Tetris [2] and Llima's Xtris [3], you will find they act quite cautiously. Here Fahey's and Llima's programs are mentioned just because they are good at algorithm visualization. We use the word cautiously because they usually eliminate lines one by one, or, once they get a chance for elimination, they just take it, without caring anything else.

However, we human DO NOT play Tetris like that AT ALL. In real Tetris game, human players tend to accumulate a few lines almost full with a single well, usually by the left or right border, then eliminate them all with one I piece. This is usually called *Tetris Action*.

As it is really boring playing Tetris cautiously, we introduce a new strategy here, which is quite similar with humankind actions. However, we would discuss the benchmark first.

A. Benchmark: counting eliminated rows ONLY is not good enough

The recent methods count eliminated rows only but nothing else. Within that benchmark, points got by eliminating lines one by one is exactly equals to points got by eliminating four lines per action. As far as we can tell, this benchmark is not fair enough to the later since the advantage of the cautious strategy is that they can survive longer.

Thus we introduce a new benchmark here as followed:

Let the number of rows can be eliminated in just one action to be x, obviously here $x \in \{1,2,3,4\}$. Then the points of each line eliminated in this action, noted as f(x), is x. So, the total points in this action, noted as g(x), equals $f(x) \cdot x = x^2$.

This benchmark is able to differentiate the cautious strategy and the adventurous strategy perfectly. Within this benchmark, the expectation of the average points may come to 2.5.

B. Literal instruction and 9+1 row

Now we officially introduce the adventurous strategy.

First we construct some *9+1 rows*, and then eliminate them all with one I piece, just like what human players do in real games. See Fig. 2. (a).

Here the concept of *9+1 rows* is easy to understand. If the left nine cells are occupied in a row and the most right one is empty, then this row is a *9+1 row*.

Obviously this is an ideal situation. In real games the sole well might appear by the left border or in any other column. Here we just simplify the situation.

If no I piece appears in a long time, some J piece or L piece may replace it as well. See Fig. 2. (b). Also, it is acceptable that some Tetromino falls down, eliminates some rows and leaves only *9+1 rows*. See Fig. 2. (c).

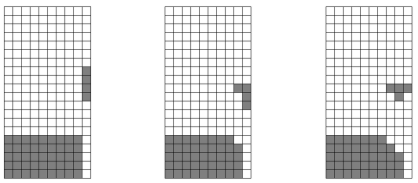

Fig. 2. (a) Ideal *9+1* with I, (b) Acceptable with J or L, (c) With T

It may looks like the sub state method by Sarjant [8], but we just look for the best fit position for I piece instead for other pieces. Also, we construct wells actively and initiatively. These are the primary differences between our method and Sarjant's.

Now we need to calculate the possibility of constructing the situation mentioned above to prove its feasibility. Considering there are only seven kinds of Tetromino, theoretically each Tetromino may appear exactly once in a round of seven appearances. Let I piece has already appeared, the next six Tetrominos may fill $6 \times 4 = 24$ cells. It is acceptable but not enough to form four *9+1 rows*, thus not able to form a *Tetris Action*. Expanding the range to 21 Tetrominos, there would be three I pieces and 18 others. Then the total height would be $21 \times 4 \div 10 = 8.4$, thus equals to the height of two I pieces placed vertically. Which means we can construct eight *9+1 rows* with 19 Tetrominos (including one I piece), and eliminate them by the other two I pieces. Therefore, it is feasible to perform this adventurous strategy.

C. It is not proper using adventurous strategy ALONE

Since the construction process of *9+1 rows* need time and high level technique, it is not proper using adventurous strategy alone. Basically, there are three typical situations as followed need to be concerned:

1) Sutiation 1: Although the incoming Tetromino sequence is random and each kind of Tetromino has equal possibility to appear each time, which is one out of seven, there is still possible that within a short period NO I appears. If we consider less than 5% as small probability, then we have the following inequality:

$$(6/7)^{20} < 5\% < (6/7)^{19}$$

Which means it is still possible to appear that a sequence longer than 20 but without a single I. As every 2.5 Tetrominos can fill a whole row, 20 Tetrominos can fill over eight rows. Since most human players have a consensus that it is very dangerous to be higher than 18, we may consider there is a security leak when higher than ten. This means, if the height is above ten, keep using adventurous strategy is just like committing suicide.

2) Situation 2: Considering the sequence, one I piece may appear in the first half. However, the ideal situation requires the I piece appears in the second half or the end of the sequence after at least four *9+1 rows* have been formed. So, if the I piece appears in an occasion not proper enough, it might be useless and have to be filled in the left nine columns.

3) Situation 3: If some holes are formed when building the *9+1 rows* and thus prevent our primary goal, which is completely unavoidable and cause I piece not able to eliminate four lines, the total height would come higher and higher.

In a word, it is definitely not safe to use pure adventurous strategy alone as it is easy to kill itself comparing with other cautious algorithms. Thus we need to introduce a new feature to solve that.

D. Danger evaluation

When human players are playing Tetris, they may change strategies dynamically. They tend to be radical when the situation is considered as safe, while be cautious when dangerous. Here the words safe and dangerous are quite abstract, but still can be considered as values of one of the features of the board, which actually is a complex feature. Here we call this abstract feature danger degree. To lower the high risk of using adventurous strategy alone, we introduce a function to evaluate the danger degree of the current board.

Since this feature indicates how dangerous the current board is, we may have to set some kind of threshold to determine whether it is too dangerous to keep using adventurous strategy. If the value of danger degree is much higher than the threshold, we may have to be as cautious as Dellacherie and Thiery to keep the AI alive.

E. Using ATM and EA to choose features and determine weights

1) Choose features: Thiery summarized all the features which have been used by several researchers in his paper [7].

In common sense, there are two kinds of philosophies in computer science: more is more and less is more. In Tetris this discipline is effective as always. Some researchers tend

to use a lot of features, even more than 12 or 16, while the others tend to use less than 10, or even only 4.

Facing this dilemma, we decided to take the less is more strategy. Our decision is based on these reasons: (a) faster and easier to evolve. (b) Dellacherie and Thiery use less attributes and get better results.

Basically referring to Dellacherie and Thiery's work, we decided to use the following eight kinds of features, most of which can be found in Thiery's paper [7]:

(a) landing_height, (b) eroded_piece_cells,

(c) row_transition, (d) column_transition,

(e) number_of_holes, (f) total_depth,

(g) total_hole_depth and (h) number_of_rows_with_hole.

We would use the other four features in some other situation, which would be referred later.

2) Cut up intervals by ATM: In previous methods, values of all the features are calculated directly, usually in a linear function. However, human players never act like that.

For example, situation (a) peak is 2; (b) peak is 8; (c) peak is 18. In previous methods, the weight of peak is constant. However, we believe it in (a) should be lower and in (c) should be higher, indicating it is very important and urgent when the peak is 18, while it is not that important when the peak is 2. It matches the real strategy which human players tend to take. Other features, such as holes, also have the same attribute.

By now we cut the whole interval into three parts. In general, we set two thresholds, noted as X_1 and X_2, and cut the intervals by the following inequality:

$$\begin{cases} x < X_1 & \text{safe, may slightly lower the weights} \\ X_1 \le x < X_2 & \text{normal, keep the originally weights} \\ x \ge X_2 & \text{dangerous, should properly raise the weights} \end{cases}$$

Still using peak height to illustrate this, we cut the height interval, which is from 0 to 20, into three parts: part 1: from 0 to 8, part 2: from 9 to 14, and part 3: from 15 to 20. Intervals of other features can be cut the same way.

In the viewpoint of ATM, we build some qualitative mapping and map the raw value to some kind of attribute and later calculate by the attribute instead of the raw value.

3) Determine weights by EA: Since we have introduced some new features and thresholds, here comes a new big issue: how can we determine their weights?

Dellacherie and Fahey chose weights by hands, based on a lot of statistical results and experience. We may not have that advantage. Since Böhm achieved fairly good results by using evolutionary algorithm, we may use that as well to generate some new weights. However, evolutions take quite a long time to perform so we decided to use weights in Thiery's algorithm as much as possible, and determine the others by ourselves to shorten the time cost.

We at least have to determine these features and thresholds:

(a) *9+1 rows*. In our method, this feature should be prominent, so we may aptly set its weight a little higher than normal to encourage the AI to generate more of them;

(b) right_column_height. We import this feature because the rightmost column plays a key role in the whole strategy. Once it is blocked, the total plan may be over. So we may punish any action that will increase the right column height;

(c) peak_height and covered (potential holes) for replacement in danger evaluation. Since landing_height and eroded_piece_cells are not available in danger evaluation, we use peak_height and covered as replacement. Their weights need to be reconfigured;

(d) Threshold of danger evaluation. If this threshold is set too high, the AI would be too venturesome so it will not live long. If it is set too low, the AI may be so cautious that we can not tell it from Dellacherie's and Thiery's algorithms.

As we need to emphasize the feature *9+1 rows*, we should pre-set its weight to be higher than others to make sure it not falls down to a minor position.

So, based on Thiery's algorithm, we use evolutionary algorithm to search the best weights for the above features and thresholds. Considering the Tetris sequence is quite random and the results are distributed in a large variance, we let every individual run five times and take the arithmetic mean value to be the performance of the individual. As some generations take too long to evolve once, we set a threshold to force it to stop to avoid wasting time.

IV. Results and analysis

A. Platform and verification

We construct an experiment platform based on Python[1] and pygame library[2]. It looks like Fahey's Standard Tetris but it is written by pure Python so the speed is a bit lower. See Fig. 3. This platform is able to change algorithms dynamically as well.

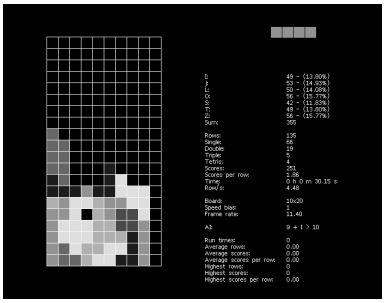

Fig. 3. The Experiment Platform

First we implement Dellacherie's and Thiery's algorithms in our platform to verify its feasibility. It turns out they perform fairly well. Then we implement our own algorithm.

With our algorithm, we use three ideal situations as followed to verify it:

(a) Pure O pieces;

(b) Pure I pieces;

(c) Mixed I and O pieces, of which the ratio is 1 by 1 and 1 by 4.

Obviously with these situations, a well designed adventurous strategy would definitely work well and keep running with

[1]See http://www.python.org for more details.

[2]See http://www.pygame.org for more details.

a low height. Especially for the second situation, it would always be like this: nine pieces of I form exactly four *9+1 rows*, then the 10th piece of I eliminate the whole thing and left nothing but an empty board behind. In fact it did, and so did the other two.

B. Experiments and results

1) Normal strategy with ATM: Following method mentioned in section III, we finally determined the intervals and respective weights by ATM and EA. The result is as followed:

F_1) landing_height: [0, 10): -13, [10, 14): -20, [14, 20]: -37

F_2) eroderd_piece_cells: -16(fixed value)

F_3) row_transition: [0, 50): -8, [50, 60): -10, [60, +∞): -14

F_4) column_transition: [0, 20): -15, [20, 30): -20, [30, +∞): -27

F_5) number_of_holes: [0, 8): -9, [8, 20): -16, [20, +∞): -26

F_6) total_depth: [0, 8): -8, [8, 12): -10, [12, +∞): -17

F_7) total_hole_depth: [0, 8): -0.87, [8, 12): -1.46, [12, +∞): -1.97

F_8) number_of_rows_with_hole: [0, 4): -19, [4, 8): -30, [8, 20]: -44

These features are noted as $F_1 \cdots F_8$, and the respective values which has been mapped from raw value to intervals are noted as $X_1 \cdots X_8$, then we have:

$$W_1 = \Sigma_{i=1}^{8} F_i X_i$$

2) Pure adventurous strategy: Since the pure adventurous strategy is only for demonstration, we simply set it as:

$$W_2 = W_1 + 100 \times nine\_and\_one\_rows$$

We did not intend to optimize the pure adventurous strategy thus this equation only emphasized the weight of *9+1 rows* on the basis of W_1.

3) Adventurous strategy with danger evaluation with ATM: we introduced the danger evaluation to keep the AI alive, although one of the indices, the average scores per row, may fall down a little.

We set the danger evaluation as:

$$\begin{aligned} W_3 = {} & \\ & 11 \times peak\_height + 9 \times row\_transition + 23 \times column\_transition \\ & + 13 \times number\_of\_holes + 6 \times covered \\ & + 24 \times number\_of\_rows\_with\_hole \\ & - 42 \times nine\_and\_one\_rows + 19 \times right\_column\_height \end{aligned}$$

in which the feature nine_and_one_rows and right_column_height are only active when the current Tetromino is I. As to whether placing the I to the rightmost column to perform a *Tetris Action* or not, we set the condition as:

$$W_4 = \begin{cases} True & nine\_and\_one\_rows \geq 4 \text{ and} \\ & \quad right\_column\_height \leq 10 \\ False & else \end{cases}$$

As to the thresholds, we set the initiatial evolutionary intervals to be (800, 1800) and (800, 1600), respectively for the current Tetromino being or not being I. After evolution, these two intervals tightened up to around (1100, 1300) and (1000, 1200).

TABLE I
COMPARED WITH THIERY'S ALGORITHM

Algorithm	**Highest Total Lines**	**Highest Total Scores**	**Average Scores per Row**
Thiery	189210	292662	1.55
Xu (Pure Adventurous)	399	953	2.39
Xu (With high thresholds: 1217, 1199)	10567	22509	2.13
Xu (With low thresholds: 1145, 1016)	68010	135246	1.99

4) Results: As we have stated, the performance of a Tetris AI is not only shown in the total lines it can eliminate, but also how it eliminates these lines, or, by what efficiency. As shown in table 1, our results are as good as Thiery's in total lines. But, while they got only 1.5 to 1.6 points on scores per row, we got 2.0 to 2.2 points, which has a 25% to 40% advantage.

C. Analysis: still not perfect

As stated in section III, our method can deal with a special Tetromino sequence of pure I. However, the way it deals with that may not be perfect. See Fig. 4. The highlighted I is number 3 in the sequence. Fig. 4. (a) is how the AI placed while Fig. 4. (b) would be what most of the experienced human players do. Notice in this figure, the dark grey I is the 3rd one, while the black cells indicate the 1st and 2nd ones, by the mean time the light grey ones are what come later.

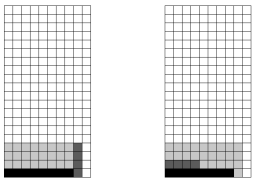

Fig. 4. (a) How AI deal with pure I sequence, (b) How human deal with it. Notice the dark grey I is the 3rd one, the black cells are placed before it and the light grey ones after

It may cause no harm as the sequence contains I pieces only. But if it does not, our AI may have made a huge mistake. However, if we calculate the weights manually, we may find it is the best plan. How could it be? We assume it is because we give the feature *9+1 rows* a too high weight. But, the key point is *9+1 rows*, if its weight goes too low, there would be no different with Thiery's algorithm.

So, we consider some mechanism of dynamically changing weights to be our next object. By that method, we may change weights dynamically according to not only its value but also its position in the whole span.

D. Analysis: well may not be always bad, but still harmful

In previous methods, number of wells and total depth of all wells are usually considered as not good at all. We assume it is not only because well is hard to be filled without I pieces but also even only one well may cause a lot of row transitions. This idea seems to have been proved by the fact that row_transition and total_depth are key features which both have high weights in almost every evaluation function.

However, in our method, well is a requirement for the *Tetris Action*. In fact, we tend to construct wells actively and initiatively. Also, in Dellacherie's and Thiery's algorithms, the feature *eroded piece cells* also tends to encourage the form of well. For example, in Thiery's algorithm, its weight is 6.6, if one I piece eliminates four lines at a time, there would be $6.6 \times 4 \times 4 = 105.6$, which is pretty high. So, we may not consider well bad simply.

But, while the platform running with our adventurous strategy, we find sometimes there are more than two wells appears. In this case, the possibility of death is much higher.

In a word, well is harmful, but not always. Take care of this feature may increase both the performance and the stability.

V. Conclusion and further improvements

The experiment may lead to the conclusion that our adventurous strategy performs as well as the former algorithms and has a 25% to 40% advantage in the index scores per row. This result is good enough but still fails to match the expectation, which as we stated is 2.5. So we may take some tactical improvements as followed:

A. Move to two pieces

The traditional methods may perform well enough by using one piece only. But, one piece is not good enough for our method.

For now we deal with I and the other six kinds of Tetrominos in quite different ways. So when we extend one piece to two pieces, we need to take care of the two following situations especially:

(a) The first or the second piece is I.

(b) The first piece is not I, but when it is placed, the third piece turns to be I.

In both cases, we may either be able to perform adventurous strategy even if we do not have proper condition, or avoid misplacing and use the late-coming I properly.

As far as we can tell, developing a two-piece algorithm with adventurous strategy would be our major future work.

B. Use J and L

As J and L can eliminate three lines a time, if we use J and L instead of I, we can improve its stability while not decreasing its performance.

L can be put in the rightmost column but as to J, the leftmost column only. As stated in Section III subsection B, we have simplified the situation. In real games, human players tend to construct the sole well in the middle area so both J and L can be placed perfectly. However, constructing a sole well in the middle is way much harder than one by the border as the *9+1 rows* are now separated into two small parts, which are much harder to be built.

C. Construct sub states for S, T and Z

As we know, S, T and Z are much harder to deal with than I, J, L and O. Sarjant's method aimed to find the auto fit position for Tetrominos, but what would happen if we construct these positions for them?

By now we construct wells only, which can be considered as sub state for I. In the last subsection, what we tend to do looks like constructing sub states for J and L. If we keep doing this, we may be able to construct sub states for S, T and Z on purpose. At that time, the performance might be much higher.

Acknowledgement

This paper is sponsored by Shanghai Maritime University Postgraduate Innovation Foundation (No. yc2011022).

References

[1] Niko Böhm, Gabriella Kókai and Stefan Mandl, *An evolutionary approach to Tetris*, 6th Metaheuristics International Conference, 2005

[2] Colin Fahey, *Tetris website*, http://www.colinfahey.com/tetris/tetris_en.html, 2011

[3] Roger Llima. *Xtris website*, http://www.iagora.com/ espel/xtris/xtris.html, 2011

[4] Yang Gao, Weiwei Wang, Xingguo Chen, et al., *Study on relational reinforcement learning*, Machine Learning and its Applications 2009: p33-48, 2009

[5] Xingguo Chen, Hao Wang, Weiwei Wang, et al., *Apply ant colony optimization to Tetris*, GECCO 09: p1741-1742, 2009

[6] Amine Boumaza, *On the evolution of artificial Tetris players*, Computational Intelligence and Games 2009, 2009

[7] Christophe Thiery and Bruno Scherrer, *Building controllers for Tetris*, International Computer Games Association Journal, 2010

[8] Samuel Sarjant, *Smart agent–creating reinforcement learning Tetris AI (Interim report)*, 2008

[9] Jiali Feng, *Attribute network computing based on qualitative mapping and its application in pattern recognition*, Journal of Intelligent and Fuzzy System, 19(2008): p243-258, 2008