Building Controllers for Tetris

Christophe Thiery, Bruno Scherrer

BUILDING CONTROLLERS FOR TETRIS

Christophe Thiery[1] *and Bruno Scherrer*[2]

Vandoeuvre-lès-Nancy Cedex, France

ABSTRACT

This article has two purposes: a review on the problem of building a controller for the well-known video game Tetris, and a contribution on how to achieve the best performance. Key components of typical solutions include feature design and feature-weight optimization. We provide a list of all the features we could find in the literature and in implementations, and mention the methods that have been used for weight optimization. We also highlight the fact that performance measures for Tetris must be compared with great care, as (1) they have a rather large variance, and (2) subtle implementation choices can have a significant effect on the resulting scores. An immediate interest of this review is illustrated. Straightforwardly gathering ideas from different works may lead to new ideas. We show how we built a controller that outperforms the previously known best controllers. Finally, we briefly discuss how this implementation allowed us to win the Tetris-domain prize of the 2008 Reinforcement Learning Competition.

1. INTRODUCTION

Tetris is a popular video game created in 1985 by Alexey Pajitnov. The game is played on a 10×20 grid where pieces of different shapes fall from the top (see Figure 1). The player has to choose where each piece is added: he can move it horizontally and rotate it. When a row is filled, it is removed and all cells above it move one row downwards. The goal is to remove as many lines as possible before the game is over, that is when there is not enough space remaining on the top of the pile to put the current new piece. A nice detailed specification of Tetris can be found on Fahey's website (Fahey, 2003).

In this contribution we give a review of works that attempt to build Tetris controllers and provide references of the feature functions involved in those works. To our knowledge, no such review exists in the literature. For a human player, the main difficulty of the Tetris game is the fact that the pieces may fall quite fast from the top of the playing area: the small amount of time often makes the decision process hard. This dimension of the problem

[1]LORIA - INRIA Lorraine, Campus Scientifique BP 239, 54506 Vandoeuvre-lès-Nancy Cedex, FRANCE. Email: thierych@loria.fr
[2]scherrer@loria.fr

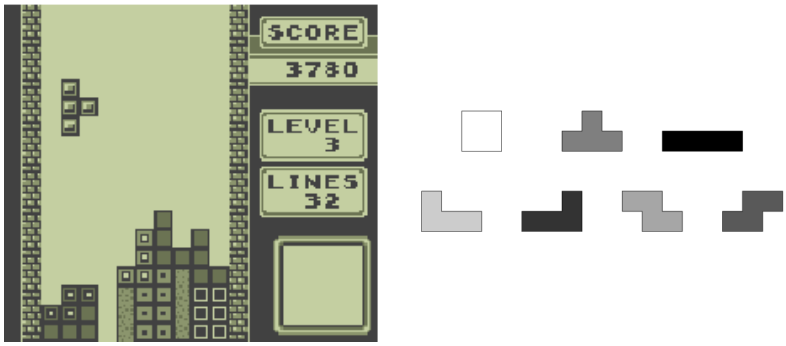

Figure 1: **Left:** a screenshot of a Tetris game. **Right:** the seven existing shapes.

does not appear when dealing with artificial players as the falling speed of the pieces is negligible compared to the decision speed of the controllers: indeed, the controllers we discuss in this article are able to play tens of thousand moves per second on a nowadays desktop computer.

The game of Tetris was chosen as a benchmark optimization problem by several researchers: the goal is to find a controller that maximizes the average number of lines, which always happens to be finite since it was shown that every Tetris game finishes with probability 1 (Burgiel, 1997). Such a problem is known to be computationally hard to solve. Tetris indeed contains a huge number (about $7.0 \times 2^{199} \simeq 5.6 \times 10^{59}$) of board configurations[3] and even in the case where the sequence of pieces is known in advance, finding the strategy that maximizes the average score is an NP-complete problem (Demaine, Hohenberger, and Liben-Nowell, 2003).

The course of the article is as follows. In Section 2 we describe the existing Tetris controllers. Section 3 reports on the difficulty of composing different Tetris controllers. In Section 4 we present our own program BCTS, that won the 2008 Reinforcement Learning Competition. BCTS is an acronym for Building Controllers for Tetris Systems. Section 5 provides a conclusion and lists two open questions.

2. EXISTING TETRIS CONTROLLERS

Most realizations of Tetris controllers (Tsitsiklis and van Roy, 1996; Bertsekas and Tsitsiklis, 1996; Kakade, 2001; Lagoudakis, Parr, and Littman, 2002; Ramon and Driessens, 2004; Farias and van Roy, 2006; Szita and Lőrincz, 2006; Llima, 2005) are *one-piece controllers*, i.e., controllers that use knowledge of the current board and the current piece only. A controller that also uses knowledge of the next piece (Fahey, 2003; Böhm, Kókai, and Mandl, 2005), as it is possible in the original Tetris setting, can make decisions that take advantage of the combination of pieces, and is called a *two-piece controller*.

All controllers rely on an *evaluation function*. In a given game *state* (i.e., (1) the current board configuration, (2) the current piece, and (3) optionally the next piece), all possible *actions* (i.e., the choice of a position and an orientation for the current piece) are evaluated by the evaluation function. Then, the controller chooses the action with the highest evaluation. Figure 2 illustrates this process. In Figure 2a, we see the principle of a one-piece controller (i.e., a one-ply evaluation process). From the current state (the current board and the current piece), each possible action is tried and leads to a new board, which is evaluated through the evaluation function. The action that leads to the best evaluation is then played. In Figure 2b, we see the principle of a two-piece controller (i.e., a two-ply evaluation process). From the current state, each possible action is tried and leads to a new board where the next piece is known. For each of these states, every possible next action is tried and the resulting boards are evaluated through the evaluation function. We then play in the first state the action leading to the state where the best evaluation was found.

In principle, it would be possible to calculate a lookahead of several stages by evaluating the actions for every possible piece at each stage and averaging the evaluations. However and as far as we know, such an in-depth search has never been carried out since it would require a huge computation time. Thus, the problem of building a Tetris controller comes down to building a good evaluation function. Ideally, this function should return high values for the good decisions and low values for the bad ones. The evaluation function is usually a combination of *feature functions*, typically (but not always; see, e.g., Böhm *et al.*, 2005) a weighted sum. Each feature function aims at capturing some relevant characteristics of the actions and states. An example of a feature is the number of holes[4] in the pile and its associated weight is usually negative: the more holes created, the lower the evaluation.

Practically designing a feature-based Tetris controller consists of two steps. The first step amounts to choosing a set of features that can extract relevant information about the game and is usually achieved by an expert. Table 2.1 provides a synthetic list of the features introduced by the works we mention herein. The second step is the tuning of the features' relative weights. In the literature, this tuning has been done manually or automatically (by reinforcement learning or some optimization techniques). We now discuss in more details what we believe are the most significant works. In Subsection 2.1 we discuss the hand-written controllers, in Subsection 2.2 the reinforcement learning approaches, and in Subsection 2.3 the general purpose optimization approaches. In Subsection 2.4 we give final comments on the best approach.

[3] This number is an estimate because it includes a few impossible configurations (see Fahey, 2003).

[4] A hole is an empty cell covered by a full cell.

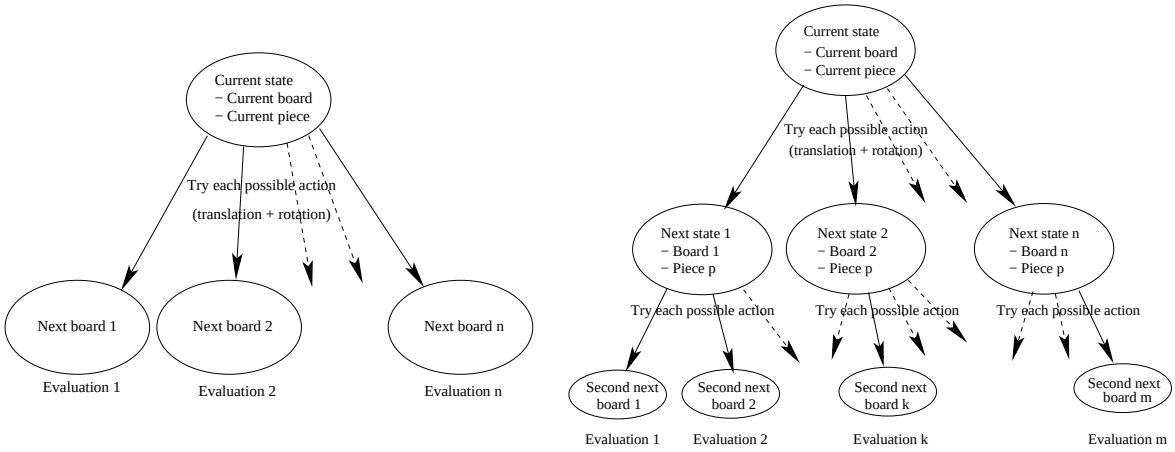

Figure 2: (a) A one-ply evaluation process. (b) A two-ply evoluation process.

2.1 Hand-written controllers

To our knowledge, the current best one-piece Tetris controller is due to Dellacherie (Fahey, 2003) and was tuned by hand. Dellacherie came up with an efficient set of features and fixed their weights manually, by trial and error. Surprisingly, this hand-written controller outperforms the other one-piece controllers from the literature, even when their weights are tuned automatically. On a total of 56 games, Dellacherie's algorithm completed an average score of about $660,000$ lines. Moreover, this measure of $660,000$ lines per game was made on an implementation of the original Tetris game, which is, as we will see in Section 3, harder than the usual simplified Tetris setting considered by most researchers. As the source code of the algorithm is freely available on Fahey's website, we reverse-engineered it and determined the features and their weights. Dellacherie's evaluation function is the following linear combination of features:

$$\begin{aligned} &-(\text{Landing height}) + (\text{Eroded piece cells}) - (\text{Row transitions}) \\ &-(\text{Column transitions}) - 4 \times (\text{Holes}) - (\text{Cumulative wells}) \end{aligned}$$

where the above features are detailed in Table 1.

Fahey (2003) created a two-piece controller with some original features of which the weights were also tuned by hand. He reports a game score of $7,200,000$ lines. Only one game was played and this took a week.

2.2 Reinforcement Learning Approaches

Several works of the reinforcement learning literature consider the game of Tetris. Some of them are mentioned in a review by Carr (2005). In the reinforcement learning context (Sutton and Barto, 1998), algorithms aim at tuning the weights such that the evaluation function approximates well the optimal expected future score from each state (recall that the state is (1) the current board configuration, (2) the current piece, and (3) optionnally the next piece). As the number of states is huge, the state space is typically visited through simulations.

The first work regarding Tetris in this domain seems to be due to Tsitsiklis and van Roy (1996). Their approach uses a feature-based Value Iteration method. The obtained controller is based on two features (the pile height and the number of holes) and makes a low average score (in the 30s). However, their work showed that using a feature-based evaluation function works better than just choosing the move that realizes the highest immediate reward (according to them, most of the time, the latter method does not score any points). Later, Bertsekas and Tsitsiklis (1996) proposed the λ-Policy Iteration algorithm and applied it to Tetris. This method generalizes the standard algorithms Value Iteration and Policy Iteration (Bellman, 1957). The evaluation function is approximated by a weighted sum of a more elaborate set of features and is estimated using simulations. They report an average score of 3,200 lines on 100 games played. Two other reinforcement learning works reused their set of features. In the first one, Kakade (2001) applied a natural policy gradient method and reported an average score of about 6,800 lines per game, without specifying how many game scores are averaged though. In the second one, Farias

Feature	**Description**	Tsitsiklis *et al.*, 1996	Bertsekas *et al.*, 1996	Llima, 2005	Lagoudakis *et al.*, 2002	Fahey, 2003	Dellacherie (Fahey, 2003)	Ramon *et al.*, 2004	Böhm *et al.*, 2005	Thiery *et al.*, 2009
Max height	Maximum height of a column	×	×	×	×			×	×	
Holes	Number of empty cells covered by a full cell	×	×	×	×	×	×	×	×	×
Column height	Height of each column		×					×		
Column difference	Height difference between each pair of adjacent columns		×					×		
Landing height	Height where the last piece is added			×			×		×	×
Cell transitions	Number of full to empty or empty to full cell transitions			×						
Deep wells	Sum of well depths, except for wells with depth 1			×						
Embedded holes	Sort of weighted sum of holes (not precisely documented)			×						
Height differences	Sum of the height differences between adjacent columns				×					
Mean height	Mean height of columns				×			×		
Δ max height	Variation of the maximum column height				×					
Δ holes	Variation of the hole number				×					
Δ height differences	Variation of the sum of height differences				×					
Δ mean height	Variation of the mean column height				×					
Removed lines	Number of lines completed at the last move				×	×			×	
Height weighted cells	Full cells weighted by their height					×			×	
Wells	Sum of the depth of the wells					×		×	×	
Full cells	Number of occupied cells on the board					×			×	
Eroded piece cells	(Number of rows eliminated in the last move) × (Number of bricks eliminated from the last piece added)						×			×
Row transitions	Number of horizontal cell transitions						×		×	×
Column transitions	Number of vertical cell transitions						×		×	×
Cumulative wells	$\sum_{w \in \text{wells}} (1 + 2 + \cdots + \text{depth}(w))$						×			×
Min height	Minimum height of a column							×		
Max − mean height	Maximum column height − Mean column height							×		
Mean − min height	Mean column height − Minimum column height							×		
Mean hole depth	Mean depth of holes							×		
Max height difference	Maximum difference of height between two columns								×	
Adjacent column holes	Number of holes, where adjacent holes in the same column count only once								×	
Max well depth	Maximum depth of a well								×	
Hole depth	Number of full cells in the column above each hole									×
Rows with holes	Number of rows having at least one hole									×
Pattern diversity	Number of different transition patterns between adjacent columns									×

Table 1: Features mentioned in Tetris publications.

and van Roy (2006) applied a linear programming approach, reaching an average score of 4,700 lines on 90 games played. Lagoudakis *et al.* (2002) applied the Least-Squares Policy Iteration method with some original features and reported an average score of 1,000 to 3,000 lines. This algorithm collects samples only once at the beginning and they show interesting convergence properties compared to the λ-Policy Iteration method. Even though it did not lead to very good performances (about 50 lines per game on average), we should also mention for completeness the work by Ramon and Driessens (2004) using relational reinforcement learning.

2.3 General Purpose Optimization Approaches

An alternative to reinforcement learning for tuning the weights is to use general purpose optimization, where an algorithm looks directly for weights that make the corresponding controller perform well, instead of trying to approximate the optimal expected score. Contrary to reinforcement learning, the resulting evaluation function does not have any specific semantics. For instance, the bot of the GNU implementation XTRIS involves six features of which the relative weights have been tuned through a genetic algorithm (Llima, 2005). The algorithm made evolve 50 sets of coefficients on 18 generations, during 500 machine hours distributed among 20 workstations. On a simulator which is very close to the original Tetris game, its author reports an average of $50,000$ lines per game. Böhm *et al.* (2005) also report interesting results, with an evolutionary approach that attemps to optimize a controller with features from the literature and several original ones. However, their results cannot be compared to most of the other works since they only consider two-piece controllers. Furthermore, for running time reasons, they do not give any averaged results on the 10×20 board. Recently, Szita and Lőrincz (2006) applied the cross-entropy method (see de Boer *et al.*, 2004), a method close to evolutionary algorithms where a population of controllers evolves around a Gaussian distribution. They used the features of Bertsekas and Tsitsiklis (1996) and reported an average score of $350,000$ lines on 30 games, outperforming the reinforcement learning approaches that used the same features.

2.4 Final comments of the best approach

This overview of the state of the art leads to a few comments. On the one hand, the features of Dellacherie seem to be the most competitive. Even with hand-chosen weights, Dellacherie's controller has so far given the best results. On the other hand, the optimization algorithms (Szita and Lőrincz, 2006; Böhm *et al.*, 2005; Llima, 2005) appear to be the most successful methods for tuning the weights of a given set of features for Tetris. The reason why reinforcement learning approaches (which try to exploit the optimal control structure of the Tetris problem) do not lead to good results is probably that for the state-of-the-art techniques of this domain, the problem of Tetris is currently too hard. The optimal score might be too difficult to estimate with a linear architecture.

3. ON THE DIFFICULTY OF COMPARING DIFFERENT TETRIS CONTROLLERS

We have mentioned the average scores reported by the authors of several Tetris controllers. This section will emphasize the fact that comparing various Tetris controllers, especially when their performances are measured on different implementations, is a difficult matter. First, we will highlight (in 3.1) the fact that the game specifications often differ from one work to another. Then, we will argue (in 3.2) that the average score of a Tetris controller has a very large deviation and we will explain how to derive confidence intervals. Eventually, we will show (in 3.3) that the performance of a controller can vary significantly because of some subtle details.

3.1 Tetris problem variations

We have already mentioned that some works propose one-piece controllers while some others consider two-piece controllers. Fahey, who made a two-piece controller, explains that when the next piece is unknown, the corresponding one-piece controller of his algorithm realizes bad scores compared to other one-piece controllers. This suggests that knowing the next piece improves considerably the performance. Conversely, it suggests that works on one-piece controllers are significantly under-rated with respect to the few works (Fahey, 2003; Böhm *et al.*,

2005) on two-piece controllers[5].

When one looks at the Tetris problem as considered by most researchers, one can observe that it is slightly different from the original game as specified by Fahey (2003). Some simplifications are often made in order to focus on the main matter of an automatic player, which is choosing a position and an orientation for the current piece. In the original game (that is as specified by Fahey, 2003), the current piece appears inside the playing area and falls gradually. The game is over when the piece has not sufficient space to appear inside the top part of the area. Most realizations (Tsitsiklis and van Roy, 1996; Bertsekas and Tsitsiklis, 1996; Kakade, 2001; Lagoudakis *et al.*, 2002; Ramon and Driessens, 2004; Farias and van Roy, 2006; Szita and Lőrincz, 2006), as well as our own implementation, consider the following simplified setting: the controller only decides in which column and orientation it drops the piece. Doing so, the game is slightly simplified since the piece does not appear in the playing area until the controller has decided where to put it. This is as if there were always sufficient space above the playing area to set the orientation and determine the column where one drops the current piece. It makes an important difference when the entire space of the board becomes available, including the top-most rows. Moreover, we prefer to avoid situations where a piece cannot move from one side of the board to the other because the pile is too high. The simplified Tetris game is easier than the original game, and a controller is likely to complete more rows[6].

3.2 The large deviation in Tetris scores

Surprisingly, though most authors seem to be aware of the large deviation of the Tetris performances, almost none of them provide confidence intervals. To our knowledge, the work by Szita and Lőrincz (2006) is the only publication that provides such confidence intervals. Below we explain how to derive them.

Fahey (2003) conjectured that the score of a Tetris game for a fixed strategy follows an exponential distribution (2003)[7]. In fact, as the score (the number of lines) is an integer, a more reasonable and qualitatively close conjecture would be that the score follows a geometric distribution. An intuition behind this conjecture is the following. Roughly speaking, the maximum height of the wall during a game resembles a one-dimensional random walk: it goes up and down depending on the random pieces; it sometimes leads back to the initial empty board. The eventual score is strongly correlated to the duration of the game, which is the time when the random walk hits the top of the board. This time, a random variable which is usually called the hitting time, is asymptotically known to be equivalent to a geometric law[8].

Even though geometric would be a better guess than exponential, Fahey's conjecture was confirmed experimentally for many controllers (each of these controllers induces an exponential distribution of scores at 95%) using the Kolmogorov-Smirnov statistical test by Szita and Lőrincz (2006). Under this conjecture (or under the geometric conjecture, respectively), the standard deviation of the score is equal (or very close, [9] respectively) to its expected value (Billingsley, 1995). Such a fact is of practical interest since it allows to assess a confidence interval. When evaluating the average score by playing several games, standard statistical analysis tells that the confidence one can have on the estimate grows with the number of games and decreases when the standard deviation grows. For a given confidence level, a larger standard deviation requires a larger number of games played. Since for Tetris the standard deviation is equal to the score, the better the controller, the harder its precise evaluation.

More precisely, a confidence interval has the following form (see any probability textbook, for instance, Billingsley (1995) for a general introduction): with some probability p, the difference between the average $\hat{\mu}$ through N

[5] Recall that a one-piece controller can be easily extended to a two-piece controller setting (see Figure 2).

[6] Note that with this common simplification, it becomes impossible to fill a "hole" by letting the current piece fall and then moving it horizontally. However, this does not make any difference because the controllers implemented with respect to Fahey's specification of the original game (Fahey, 2003) do not exploit this ability either.

[7] This is a typical property of the Tetris game. There is no reason to believe that other games have the same kind of property.

[8] This can be seen as a consequence of the well-known Perron-Froebenius Theorem (Billingsley, 1995): let λ be the biggest (in module) non-1 eigenvalue of the stochastic matrix associated with the random walk. When the time t tends to infinity, the probability that the random walk is still in a non-absorbing state at time t is equivalent to $a|\lambda|^t$ for some constant a. As a consequence, the probability that the hitting time is exactly t, is equivalent to $a|\lambda|^{t+1} - a|\lambda|^t$, which is proportional to $|\lambda|^t$.

[9] A geometric law of parameter p has mean $\frac{1}{p}$ and standard deviation $\sqrt{\frac{1-p}{p^2}}$ (Billingsley, 1995). When p is very small (this the case when the Tetris controllers are good), the standard deviation is equivalent to $\frac{1}{p}$.

games of the expected Tetris score μ with standard deviation $\sigma = \mu$ satisfies:

$$|\mu - \hat{\mu}| \leq \frac{k\sigma}{\sqrt{N}} = \frac{k\mu}{\sqrt{N}} \simeq \frac{k\hat{\mu}}{\sqrt{N}}$$

where k is a constant that depends on the chosen probability p (typical relations are $k = 1$ for $p = 0.68$, $k = 2$ for $p = 0.95$, $k = 3$ for $p = 0.997$). Equivalently, this leads to the following *relative* confidence interval:

$$\frac{|\mu - \hat{\mu}|}{\hat{\mu}} \leq \frac{k}{\sqrt{N}}.$$

As an illustration, consider the evaluation of Dellacherie's controller with $N = 56$ games. The above analysis implies that the confidence of the empirical average ($660,000$ lines) is $\pm 27\%$ with probability 0.95. Despite its large size, this confidence interval confirms the fact that, with high probability (0.95), no publicly known one-piece algorithm does better. More generally, controllers evaluated with $N = 100$ games lead to a confidence of $\pm 20\%$ that is valid 95% of the time. In the rest of the paper, we chose to use the notation $m \pm c\%$ to represent confidence intervals valid 95% of the time (corresponding to $k = 2$). We chose this value of 0.95 in order to provide intuitive confidence intervals for Tetris scores, that are valid most of the time.

3.3 Tetris implementations subtleties

We have just seen that, in general, the confidence intervals that we can derive for Tetris controllers are large. When we implemented our own Tetris simulator, we further noticed that some subtle details in the implementation could have a significant effect on the performance measures. We here discuss these details and their influence on the game scores, using the example of Dellacherie's controller.

A first subtle detail (which is never explicitly mentioned in the publications we have seen so far) is how an evaluation-based controller acts when it is close to game over. It might be that the decision that has the highest evaluation leads to game over while other decisions (with lower evaluations) do not. In such a case, it is better not to consider actions that directly lead to game over: those actions will not be evaluated, the game will last longer and the score will be better. If we act this way, the game is over if and only if *all* decisions lead to game over. We implemented Dellacherie's algorithm this way, and it completes $5,200,000 \pm 20\%$ rows on average. If we let the controller evaluate actions that lead to game over, Dellacherie's performance drops to $850,000 \pm 20\%$ rows with our implementation.

Additionally, we believe that most implementations of Tetris define game over as the moment when the current piece cannot fit into the board, that is when the current piece overflows the 10×20 board. However, if we closely examine the description of Tetris considered by Bertsekas and Tsitsiklis (1996), we can see they consider that the game *"ends when a square in the top row becomes full and the top of the wall reaches the top of the grid"*. This latter definition is equivalent to saying that the pile overflows in a 10×19 board. Such a subtle detail can make a significant difference on the game scores: with a 10×19 board, Dellacherie's algorithm completes $2,500,000 \pm 20\%$ rows instead of $5,200,000 \pm 20\%$. For this reason, we actually believe that the experimental results of Bertsekas and Tsitsiklis that we mentioned in Section 2 are underrated with respect to the other reinforcement learning works.

As small details concerning the game rules and the controller implementations can have significant effects on the game scores, particular care is required when comparing different approaches. Most of the results reported by previous works are actually not comparable since they use different Tetris implementations. In particular, when Dellacherie's score has to be compared to scores realized with the simplified Tetris setting (considered by most researchers, see above), it should be considered as $5,200,000 \pm 20\%$ rows instead of $660,000 \pm 27\%$, which was its reported score on an original Tetris simulator[10]. The only way to do a fair comparison of various controllers is to run them on the same simulator and a large number of times. To this end, we have implemented a configurable and optimized Tetris simulator, and several controllers[11].

[10] Although the score of $660,000 \pm 27\%$ was obtained with a more restrictive implementation than the usual simplified setting, it was already the best performance known.

[11] The C source code is available here: `http://gforge.inria.fr/projects/mdptetris`.

4. THE 2008 REINFORCEMENT LEARNING COMPETITION

Based on this careful analysis of the Tetris literature, we implemented a controller that combines previous successful feature design and feature-weight optimization. We called the program BCTS, which stands for Building Controllers for Tetris Systems. We trained a controller by the cross-entropy method (Szita and Lőrincz, 2006), using the performant features of Dellacherie (Fahey, 2003) and two original features mentioned in Table 2.1: the hole depth and the number of rows with holes. We give here the evaluation function we obtained after the cross-entropy optimization of the weights:

$$-12.63 \text{ (Landing height)} + 6.60 \text{ (Eroded piece cells)} - 9.22 \text{ (Row transitions)} - 19.77 \text{ (Column transitions)}$$
$$-13.08 \text{ (Holes)} - 10.49 \text{ (Cumulative wells)} - 1.61 \text{ (Hole depth)} - 24.04 \text{ (Rows with holes).}$$

On the simplified Tetris setting considered by most researchers (see Section 3.1), this controller achieves an average score of $35,000,000 \pm 20\%$ lines with probability 0.95. Although we did not test it on the real Tetris game (as specified by Fahey, 2003), a conservative lower bound on the score it would realize is $910,000 \pm 5\%$ lines with probability 0.95. We obtained this bound by playing games on a 10×16 board of the simplified setting; indeed, any move possible on the 10×16 board with the simplified setting would also be possible on the 10×20 board with the real Tetris game since the height of all Tetris pieces is lower than or equal to 4. More details regarding how we built BCTS can be found in Thiery and Scherrer (2009).

Using the bibliographical work we present in this article, BCTS recently won the 2008 Reinforcement Learning Competition. This competition made Tetris controllers play on modified instances of the problem, where some properties of the game (for instance, the board size and the scoring function for making 1-2-3 or 4 lines) could vary and the artificial players had to adapt themselves to each environment. The performance measure [12] that was used to compare the different players had much less variance than the natural measure which we discussed in this article. Though the problem was formulated in the reinforcement learning context for this competition, any kind of method was allowed. We used a modified version of our controller discussed above, with an additional original feature called "pattern diversity", which looks at the pattern formed by the top part of two adjacent columns and counts how many different patterns are present. This encourages the controller to ensure that the board can absorb any piece without making a hole. We know that the artificial players that obtained the second and the third places (personal communication with Marek Petrik and István Szita respectively) were also tuned by the cross-entropy method. We believe that our choice of features was decisive to win the competition.

5. CONCLUSION AND OPEN QUESTIONS

To our knowledge, this article is the first in-depth review on the problem of building a Tetris controller, summarizing the most significant realizations (hand-written, reinforcement learning, general optimization) and providing a list of the feature functions. This is meant to be a working base for anyone interested in the Tetris problem. Furthermore, we showed that comparing results from different implementations may not make sense because small differences have a significant effect on game scores. This warning may be applied to other games than Tetris too.

5.1 Two open questions

An observation of this review is that optimization approaches like the cross-entropy method (Szita and Lőrincz, 2006) or the evolutionary methods (Böhm *et al.*, 2005; Llima, 2005) have been much more successful so far than reinforcement learning approaches. However, the latter provide nice theoretical tools to calculate in particular the expected score (the *value function*) of the best Tetris strategy. Thus, we were actually able to build the exact optimal player for a reduced instance of Tetris (a 5×5 board) using the Value Iteration algorithm (Puterman, 1994; Sutton and Barto, 1998). On this reduced board, the expected value of the score of the optimal strategy is 13.7 lines. With the real size of the Tetris board, even if the reinforcement learning algorithms suffer from the curse of dimensionality and have difficulties to estimate the future score when approximation is needed, an advantage is that they estimate future scores instead of just trying to maximize them. Although the optimization methods such as cross-entropy and genetic algorithms perform better on Tetris, they provide no information

[12] The performance was the total score after a fixed number of interactions, with no penalty for losing a game (except a reset to the initial empty board).

about the optimal possible score. This leads us to our first open question. It would be interesting to continue investigations in this direction in order to estimate an upper bound on the expected value of the optimal player's score owing to the reinforcement learning framework.

A second natural question that remains open after this review is to determine what optimization method is the most suitable for Tetris. The state-of-the-art optimization approaches to tune the weights (Llima, 2005; Böhm *et al.*, 2005; Szita and Lőrincz, 2006) cannot be compared directly because they use different implementations and different features. Furthermore, contrary to the two other works and the reinforcement learning approaches, Böhm *et al.* (2005) only consider two-piece controllers. Hence, their performance compared to one-piece controllers is still unknown. It would be interesting to implement and execute those methods under the same conditions, to determine in what circumstances the cross-entropy method can perform better than genetic algorithms (if it does), and whether such observations would be specific to Tetris or could apply to other problems as well.

6. REFERENCES

Bellman, R. E. (1957). *Dynamic Programming.* Princeton University Press, Princeton, NJ.

Bertsekas, D. and Tsitsiklis, J. (1996). *Neurodynamic Programming.* Athena Scientific.

Billingsley, P. (1995). *Probability and measure.* John Wiley & Sons, New York, N.Y., 3rd edition.

Boer, P. de, Kroese, D., Mannor, S., and Rubinstein, R. (2004). A tutorial on the cross-entropy method. *Annals of Operations Research*, Vol. 1, No. 134, pp. 19–67.

Böhm, N., Kókai, G., and Mandl, S. (2005). An Evolutionary Approach to Tetris. *The Sixth Metaheuristics International Conference (MIC2005).*

Burgiel, H. (1997). How to Lose at Tetris. *Mathematical Gazette*, Vol. 81, pp. 194–200.

Carr, D. (2005). Applying reinforcement learning to Tetris. Technical report, Computer Science department of Rhodes University.

Demaine, E. D., Hohenberger, S., and Liben-Nowell, D. (2003). Tetris is hard, even to approximate. *Proc. 9th International Computing and Combinatorics Conference (COCOON 2003)*, pp. 351–363.

Fahey, C. P. (2003). Tetris AI, Computer plays Tetris. `http://colinfahey.com/tetris/tetris_en.html`.

Farias, V. and Roy, B. van (2006). *Tetris: A study of randomized constraint sampling.* Springer-Verlag.

Kakade, S. (2001). A natural policy gradient. *Advances in Neural Information Processing Systems (NIPS 14)*, pp. 1531–1538.

Lagoudakis, M. G., Parr, R., and Littman, M. L. (2002). Least-squares methods in reinforcement learning for control. *SETN '02: Proceedings of the Second Hellenic Conference on AI*, pp. 249–260, Springer-Verlag, London, UK.

Llima, R. E. (2005). Xtris readme. http://www.iagora.com/~espel/xtris/README.

Puterman, M. (1994). *Markov Decision Processes.* Wiley, New York.

Ramon, J. and Driessens, K. (2004). On the numeric stability of gaussian processes regression for relational reinforcement learning. *ICML-2004 Workshop on Relational Reinforcement Learning*, pp. 10–14.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning, An introduction.* BradFord Book. The MIT Press.

Szita, I. and Lőrincz, A. (2006). Learning Tetris Using the Noisy Cross-Entropy Method. *Neural Computation*, Vol. 18, No. 12, pp. 2936–2941.

Thiery, C. and Scherrer, B. (2009). Construction d'un joueur artificiel pour Tetris. *Revue d'Intelligence Artificielle*, Vol. 23, pp. 387–407.

Tsitsiklis, J. N. and Roy, B. van (1996). Feature-Based Methods for Large Scale Dynamic Programming. *Machine Learning*, Vol. 22, pp. 59–94.