Tetris Agent Optimization Using Harmony Search Algorithm

Victor II M. Romero[1], Leonel L. Tomes[2] and John Paul T. Yusiong[3]

**[1,2,3]Division of Natural Sciences and Mathematics,
University of the Philippines Visayas Tacloban College
Tacloban City, Leyte, 6500, Philippines**

Abstract

Harmony Search (HS) algorithm, a relatively recent meta-heuristic optimization algorithm based on the music improvisation process of musicians, is applied to one of today's most appealing problems in the field of Computer Science, Tetris. Harmony Search algorithm was used as the underlying optimization algorithm to facilitate the learning process of an intelligent agent whose objective is to play the game of Tetris in the most optimal way possible, that is, to clear as many rows as possible. The application of Harmony Search algorithm to Tetris is a good illustration of the involvement of optimization process to decision-making problems. Experiment results show that Harmony Search algorithm found the best possible solution for the problem at hand given a random sequence of Tetrominos.

Keywords: *Harmony Search algorithm, Tetris, Intelligent Agent, Artificial Intelligence*

1. Introduction

Problems and challenges have always been part of human life and of the human civilization itself. They define the difference between what is currently in existence and of what could be, after a goal has been achieved. In Computer Science, researchers are concerned with the search for solutions to computational problems and these problems may be categorized into two main classes: P-problems and NP-problems.

P-problems, otherwise known as Polynomial-time problems are problems whose solutions may easily be identified, that is, the procedure for finding the solution is already known. On the other hand, NP-problems are problems whose solutions have no proven optimal way of acquisition. NP-problems are also called "*I know it when I see it problems*" because of the fact that the validity of their solution may only be verified when tried and evaluated [4,5]. A good example of such problem is the creation of an intelligent agent for Tetris [5].

Tetris is a puzzle computer game originally created by Alexey Pajitnov [6]. An intelligent agent for Tetris is a program whose goal is to be able to play the game in the most optimal way possible. In such a case, we only know the quality of the agent by assessing its performance when it has already played the game. To deal with such problems, where too little detail is known on the nature of a problem's solution, computer scientists use a different approach in the form of meta-heuristic algorithms.

Meta-heuristic algorithms are a primary sub-field of a larger class of algorithms and techniques called stochastic optimization [4]. They are called stochastic optimization because they employ some degree of randomness in searching for a solution. In other words, they are solving problems through a series of intelligent guesses. The primary ideas for achieving the series of intelligent guesses of existing meta-heuristic algorithms are driven by natural occurrences like biological processes and animal behaviors.

The popularity of Tetris has intrigued mathematicians and computer scientists to study its non-trivial nature and reveal its NP-Complete characteristics [5] triggering the motivation for the creation of an intelligent agent. A Tetris intelligent agent is an Artificial Intelligence (AI) program which plays or simulates the game with the goal of clearing as many rows as possible. In fact, in the past years scientists have successfully created intelligent agents using Evolutionary Algorithms [1,2] and Ant Colony Optimization [3].

Harmony Search (HS) algorithm is a meta-heuristic algorithm developed in 2001 by Geem *et al* [7]. It is modeled after the musical improvisation process, wherein a band of musicians continuously tries to create better harmony. This algorithm and its variants [8-9] have been applied to a wide array of real-life optimization problems such as structural design, ecological conservation, industrial operation and musical composition [7], [10-13]. The HS algorithm is a powerful optimization tool because of its ability to discover the high performance regions of the solution space in a reasonable amount of time. In addition, other characteristics enable the HS algorithm to increase its flexibility and produce better solutions, and these are [14]:

1. HS imposes fewer mathematical requirements.

2. HS uses stochastic random searches thus any derivative information is unnecessary.
3. HS creates a new solution vector after considering all of the existing solution vectors.

Since the Harmony Search algorithm has demonstrated its strength on various fields of discipline and has been successfully applied to many problem domains, this study explores the feasibility of using the Harmony Search algorithm in decision-making optimization problems, that is, to use the HS algorithm as the underlying optimization algorithm in facilitating the learning process of the Tetris intelligent agent.

The paper is organized as follows. A brief description about Tetris is presented in Section 2. Section 3 introduces the Harmony Search algorithm followed by a discussion on the proposed HS-based Tetris intelligent agent in Section 4. The experimental results are shown in Section 5 while Section 6 contains the conclusion.

2. The Tetris Game

Tetris is a game originally invented and programmed by Alexey Pajitnov in June 6, 1984 while working in Dorodnicyn Computing Center of the Academy of Science of the USSR in Moscow [6]. It is one of the most popular and most successful games to hit the market. In fact, Tetris' success as a computer game led to the creation of many other variants, sporting slightly different game play. The Tetris game and its variants are basically composed of two main components, the game pieces called Tetrominos and the game board.

The standard Tetris game board has a dimension of 10 x 20 and there are seven Tetrominos or game pieces, as shown in Figure 1and these are O, J, L, I, S, T and Z. The game pieces differ significantly by the maximum number of rows that they are able to clear simultaneously. In fact, a simple analysis of the game pieces reveals that all are capable of clearing one and two rows. But only pieces "I", "J" and "L" are capable of clearing three rows and ultimately, only the "I" Tetromino is capable of clearing four rows (called "Tetris")[1].

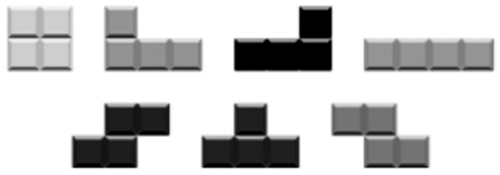

Fig. 1 Tetris Game Pieces-Tetrominos

In Tetris, Tetrominos fall from the top of the game board one at a time and aside from the current Tetromino being manipulated, an advance view of the next piece is provided to the player. This is to enable the player to manipulate and position the current Tetromino in the game board such that one or more gapless row(s) of block is created. When such a scenario happens, the gapless row is cleared and all existing blocks above that row descend *n* units, where *n* is the number of rows cleared which is between 1 and 4. In addition, manipulation of a Tetromino can only be performed in two ways, either by doing a 90 degree rotation or a sideways movement while the Tetromino has not yet reached the bottom of the game board, at which case it fixes itself into position [1].

However, unlike most games, Tetris does not have a win condition, so the game continues until the stack of blocks in the game board disallows the entry of succeeding Tetrominos. This means that the only goal in the game is to be able to clear as many rows as possible for better and longer game play. As a result, staying in the game solely depends upon the number of rows cleared.

3. The Harmony Search (HS) Algorithm

Computer scientists have found a significant relationship between music and the process of looking for an optimal solution. This interesting connection led to the creation of the Harmony Search algorithm. It is a new kind of meta-heuristic algorithm mimicking a musicians' approach to finding harmony while playing music. When musicians try to create music, they may use one or a combination of the three possible methods for musical improvisation which are as follows: (1) playing the original piece, (2) playing in a way similar to the original piece, and (3) creating a piece through random notes.

In 2001, Geem *et al* [7] saw the similarities between the music improvisation processes and finding an optimal solution to hard problems and formalized the three methods as parts of the new optimization algorithm, the Harmony Search algorithm (HS); (1) harmony memory consideration (2) pitch adjustment and (3) randomization. These three methods are the main parameters of the algorithm and play a vital role in the optimization process [7], [10-13].

For musicians, one of the ways of producing good music is considering existing compositions and playing them as they are. In the Harmony Search algorithm, this is also the case, the use of harmony memory is vital as it ensures that potential solutions are considered as elements of the new solution vector. The second way of coming up with good music is by playing something similar relative to an existing composition, in HS this is called the pitch adjustment mechanism and may be referred to as the exploitation mechanism in the Harmony Search algorithm; it is responsible for generating slightly varying solution from existing solutions. It is comparable to the mutation mechanism in genetic algorithms. Randomization, the last of the methods ensures that the search for solution is not isolated to the local optima. It makes the solution set more diverse by not limiting the search for solution in a confined area and is referred to as the exploration mechanism of the Harmony Search algorithm [7], [10-13].

So, in the Harmony Search algorithm, each musical instrument is represented as a decision variable. The value of each decision variable is set in a similar manner that a musician plays his instrument, contributing to the overall quality of the music created, thus the name Harmony Search. The pseudo-code of the Harmony Search algorithm as presented, shows that the optimization process is done on a per decision variable basis for each harmony (solution) in the harmony memory.

Furthermore, based on the pseudo-code and as shown in Figure 2, the optimization process of the Harmony Search algorithm may be described in three main steps:

1. **Initialization**: Program parameters are defined and the harmony memory is initialized by filling it up with random solutions; each harmony is evaluated using an evaluation or objective function.
2. **Harmony improvisation**: A new solution is created. The three methods of the Harmony Search algorithm are used to decide on the value that will be assigned to each decision variable in the solution.
   a. **Creation of a new solution**: A new solution is created either (1) randomly with a probability of $\mathbf{1 - r_{accept}}$ or alternatively (2) by copying an existing solution in the harmony memory, with a probability equal to r_{accept}.
   b. **Adjustment:** With a probability of r_{pa}, the elements of the new harmony are then modified.
   c. Using the objective function, the new harmony is evaluated.
3. **Selection**: When a terminating condition is met, the best harmony (solution) in the harmony memory is selected.

Also, there are several parameters that have to be defined before the start of the optimization process.

(1) **Maximum number of cycles or iterations** – is the basis for terminating the optimization process.
(2) **Harmony memory size** – refers to the number of harmonies that will be stored in the harmony memory.
(3) **Number of decision variables** – each harmony is composed of several decision variables.
(4) **Harmony Memory Consideration rate (r_{accept})** – determines the rate at which decision variables in the harmony are considered as elements of the new harmony that will be created.
(5) **Pitch Adjustment rate (r_{pa})** - defines the probability for adjusting the values of decision variables copied from an existing harmony in the harmony memory by adding a certain value.

```
Begin
 Define objective function $f(x)$, $x = (x_1, x_2 \dots x_d)^T$
 Define harmony memory accepting rate ($r_{accept}$)
 Define pitch adjusting rate ($r_{pa}$) and other parameters
 Generate Harmony Memory (HM) with random harmonies
    While (t < max number of iterations)
      While (i <= number of variables)
        If(rand < $r_{accept}$)
             Choose a value from HM for the variable i
             If(rand < $r_{pa}$)
               Adjust the value by adding certain amount
             End if
        Else
             Choose a Random Value
        End if
      End while
      Accept the new harmony (solution) if better
    End while
    Find current best solution
End
```

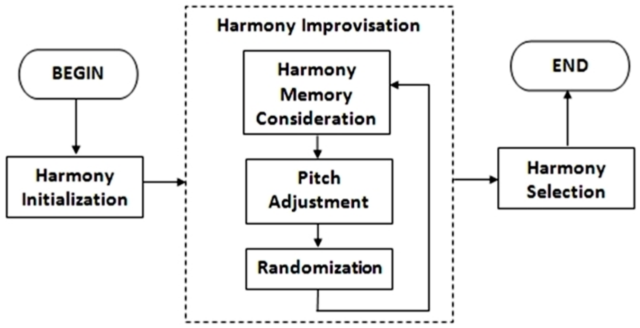

Fig. 2 The Harmony Search Optimization Process

4. HS-based Tetris Intelligent Agent

When a human first plays a Tetris game, he/she may play the game with no more than the goal of clearing as many rows as possible. However, as the game progresses or as he/she continues to play the game repeatedly, it becomes obvious that like every decision-making process in life, in choosing the best move for a game piece in Tetris, there are several factors to put into consideration for the maximization of the number of rows cleared.

Through the efforts of previous researches [1-3], [5], these factors were transformed into forms that can be implemented in computers and are called feature functions. Also, similar to the decision-making process in life, we assign weights to each of these factors to symbolize its significance or bearing to the final decision. Ultimately, the best move for a current piece is chosen using a linear (summation) combination of these feature functions with their corresponding weights, called the *state-evaluation function*, as shown in Eq. (1).

$$V(s) = \sum_{i=1}^{19} w_i f_i(s) \qquad (1)$$

where s is the state (board configuration), w_i represents each of the weights of the i^{th} feature function $f_i(s)$ and $f_i(s)$ is a function that maps a state (a board configuration) to a real value. The goal of the optimization process is to be able to find an optimal set of weights that will result in the most number of cleared rows.

The 19 feature functions identified and used in this research are as follows: (1) pile height, (2) holes, (3) connected holes, (4) removed rows, (5) altitude difference, (6) maximum well depth, (7) sum of all wells, (8) landing height, (9) blocks, (10) weighted blocks, (11) row transitions, (12) column transitions, (13) highest hole, (14) block above highest hole, (15) potential rows, (16) smoothness, (17) eroded pieces, (18) row holes, and (19) hole depth.

Thus, the solutions generated by the program will come in the form of a vector of 19 weights, that is, w_1 *to* w_{19}. Feature functions 1-12 are from [2], 13-16 are taken from [3] and 17-19 are from [1]. A description of the feature functions is presented in Section 4.1.

As illustrated in Figure 3, the proposed HS-based Tetris intelligent agent called Harmonetris, is divided into two main parts;

(1) the solution optimizer, and
(2) the Tetris game simulator.

The solution optimizer comprises the Harmony Search algorithm part of the program while the Tetris game simulator is composed of our Tetris agent as well as the Tetris game itself.

The solution optimizer generates harmonies or solutions in the form of a set of weights; normally, $w_1$to w_{19}. Each of this set of weights is passed to the Tetris game simulator, which plays one complete game of Tetris using the weights provided and a randomly generated sequence of Tetrominos. After playing the game, the game simulator returns the number of rows cleared by the agent based on the current assigned weights to the solution optimizer. The number of cleared rows serves as the objective function, where more cleared rows means a better agent performance. When a terminating condition has been met, the solution optimizer then outputs the best solution created so far.

The Tetris game simulator uses the state-evaluation function in Eq. (1) to evaluate on a per Tetromino move basis for each feature function while the objective function rates an entire Tetris game in the form of maximum number of rows (lines) cleared which is the basis for the solution optimizer to determine the objective function value of each harmony.

The main idea behind this set up is to use the Harmony Search algorithm as the solution optimizer's underlying optimization algorithm.

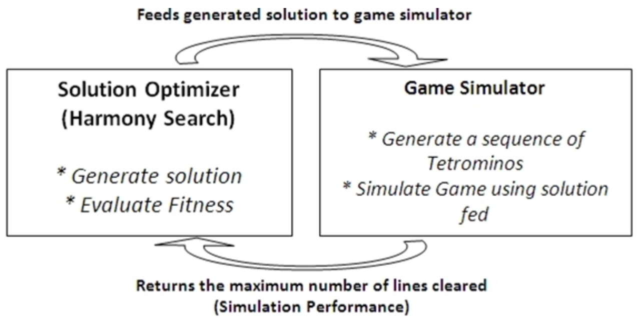
Feeds generated solution to game simulator
Solution Optimizer (Harmony Search)
* Generate solution
* Evaluate Fitness
Game Simulator
* Generate a sequence of Tetrominos
* Simulate Game using solution fed
Returns the maximum number of lines cleared (Simulation Performance)

Fig. 3 The Harmonetris Overall Program Flow

4.1 The Feature Functions

A description of each of the feature functions, f_i, used in this research is presented in this section.

1. **Pile Height:** The row of the top most Tetromino in the board. Each of the filled cells reached directly from the top are compared and the row value of the topmost filled cell is the pile height. In Figure 4 the pile height is 13.

2. **Holes:** The number of all gaps with at least one occupied cell above them. In Figure 5(a), the holes in the board are marked with "1". The number of holes in the board is 10.

3. **Connected Holes:** Similar to Holes, however, counts vertically connected gaps as one. Figure 5 shows the difference between Holes and Connected Holes. Connected Holes has a value of 7 with each connected hole in the board marked with "1".

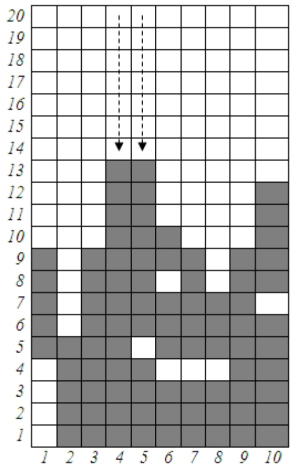
Fig. 4 The Feature Function: Pile Height

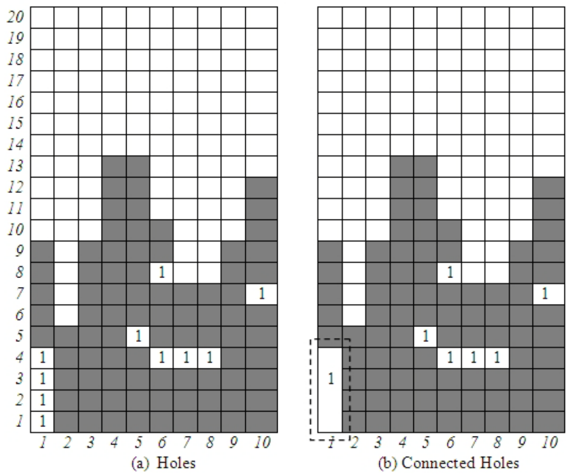

Fig. 5 Difference between Holes and Connected Holes

4. **Removed Rows:** The number of rows cleared by the last step. This is the number of rows that was cleared in order to arrive at the new board configuration.

5. **Altitude Difference:** The difference between the lowest gap directly reachable from the top and the highest occupied cell. Figure 6 has an altitude difference of 8.

6. **Maximum Well Depth**: The depth of the deepest well on the board. This is 4 in Figure 7.

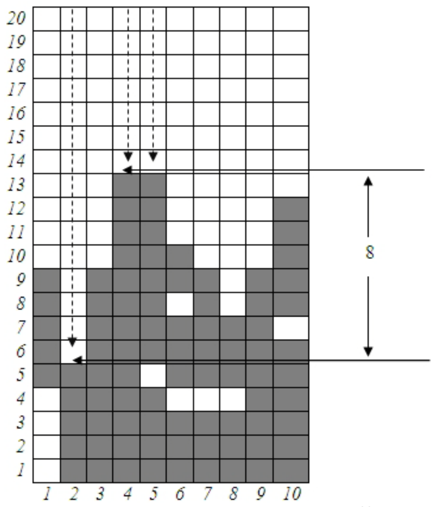

Fig. 6 The Feature Function: Altitude Difference

7. **Sum of all Wells**: The sum of all wells on the game board. The board in Figure 7 has a value of 6.

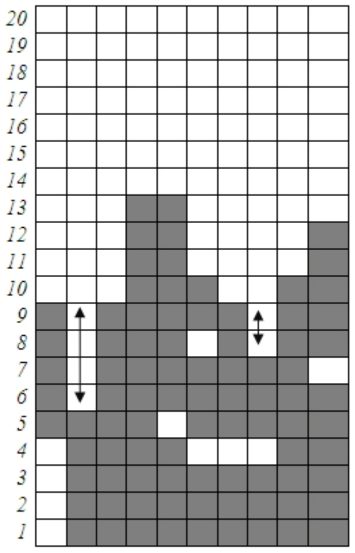
Fig. 7 The Feature Function: Sum of Wells

8. **Landing Height:** The height at which the last Tetromino has been placed. Figure 8 shows that the landing height of piece ‘I’ is 9.

9. **Blocks:** The number of cells that has been occupied in the board.

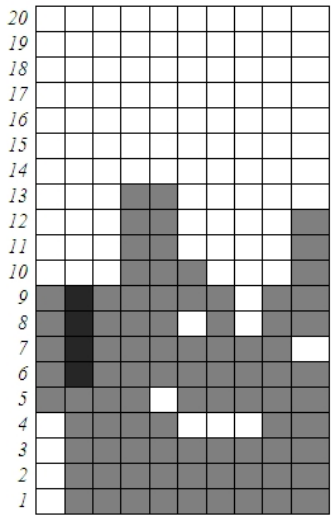

Fig. 8 The Feature Function: Landing Height

10. **Weighted blocks:** Same as Blocks, however counting from the bottom to the top, blocks located at row *n* count *n*-times as much as the blocks in row 1. Figure 9 illustrates the weight of each of the blocks.

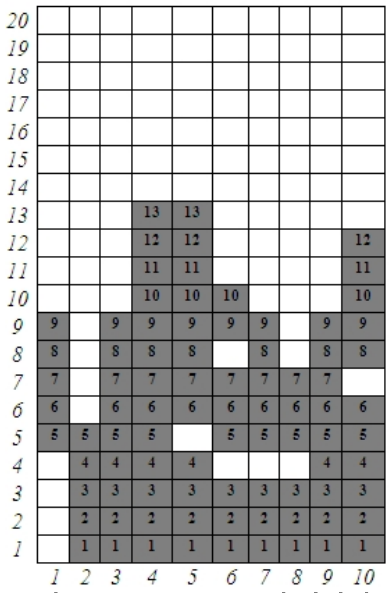

Fig. 9 The Feature Function: Weighted Blocks

11. **Row Transitions:** The sum of all occupied or unoccupied transitions. Each arrow in Figure 10 counts as one row transition.

12. **Column Transitions:** Same as Row Transitions, however, it only counts vertical transitions. Figure 11 provides a clear illustration of the column transitions in the board.

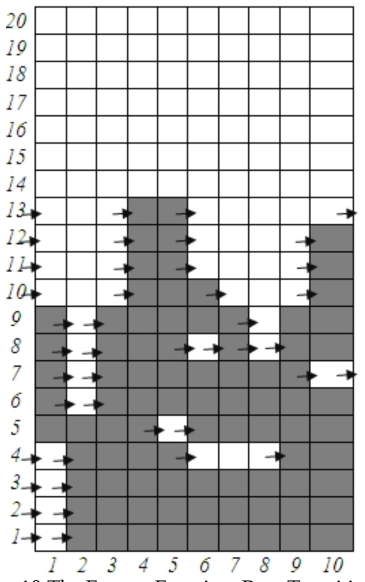

Fig. 10 The Feature Function: Row Transitions

13. **Highest Holes:** The height of the topmost hole on the game board. This is 8 in Figure 5(a).

14. **Blocks Above Highest Hole:** The number of blocks on top of the Highest Hole. In Figure 5(a) the highest hole in the board is at 8, so the number of blocks above it is 2.

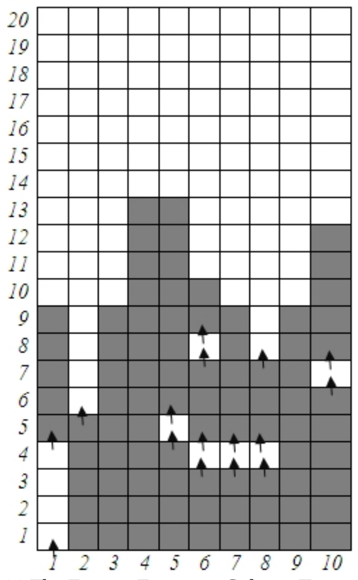

Fig. 11 The Feature Function: Column Transitions

15. **Potential Rows:** The number of rows located above the Highest Hole and in use by more than 8 cells. This is 0 in Figure 5(a).

16. **Smoothness:** The sum of all absolute differences of adjacent column height, as well as the difference of the first and last column. Using the board in Figure 11, Table 1 illustrates how the value of smoothness is computed.

Table 1: Sample Computation: Smoothness

Columns	*Column Heights*	*Value*
1, 2	|9-5|	4
2, 3	|5-9|	4
3, 4	|9-13|	4
4, 5	|13-13|	0
5, 6	|13-10|	3
6, 7	|10-9|	1
7, 8	|9-7|	2
8, 9	|7-9|	2
9, 10	|9-12|	3
10, 1	|12-9|	4
Smoothness		**27**

17. **Eroded Pieces:** The number of rows cleared in the last move multiplied with the number of cells of the last piece that were eliminated in the last move.

18. **Row Holes:** The number of rows with at least one hole. Figure 12 shows how the value for row holes is computed. In the said figure, the value is 7.

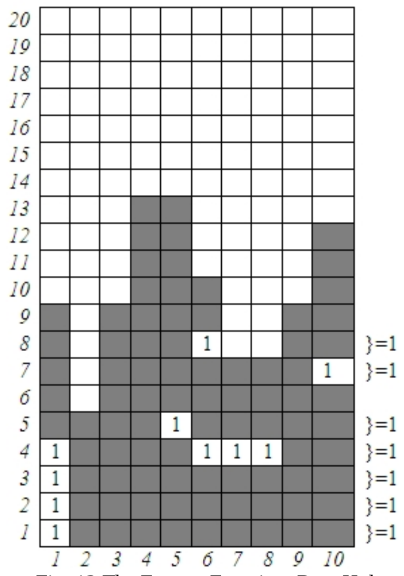

Fig. 12 The Feature Function: Row Holes

19. **Hole Depth:** The number of filled cells on top of each hole. Table 2 shows how to compute for the hole depth based on Figure 13.

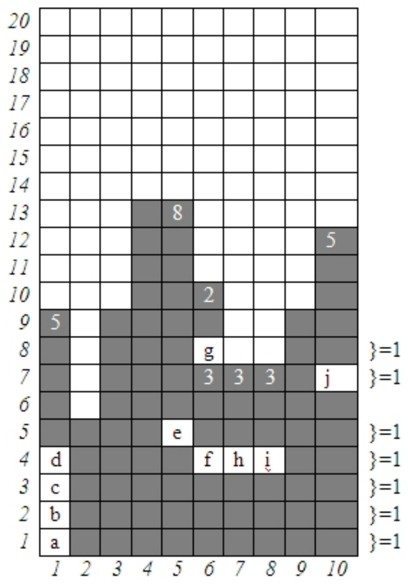

Fig. 13 The Feature Function: Hole Depth

Table 2: Sample Computation: Hole Depth

Hole	*Filled Cells*
a	5
b	5
c	5
d	5
e	8
f	3+2=5
g	2
h	3
i	3
j	5

4.2 The Tetris Game Simulator

Every generated harmony of the solution optimizer in the form of a vector of weights is fed to the Tetris Game Simulator. The simulator performs a simulation of the game using the feature functions as basis for determining the best move, and then it returns the maximum number of cleared rows, which is the *objective function value* of the harmony.

The Tetris Game Simulator has two main components: (1) Tetris Game and (2) Tetris Agent. The Tetris Game defines the logical characteristics of the game and the rules on how it must be played. The Tetris Agent plays the Tetris Game until a termination condition is satisfied and then returns the required result. The agent always selects the move that will yield the best outcome, which in turn maximizes the result. The simulation will only terminate if the current state of the game satisfies a termination condition. There are two ways to end the simulation. One is by limiting the number of pieces spawned and the other is waiting for the game to be over.

4.3 The Tetris Agent

In a Tetris Game, one piece is being played at a time. The player has to select from among the 10 possible translations (positions), that is, where to place the current piece. Also, the player may change the orientation (can be up to 4 possible orientations) of the game piece depending on how beneficial this move may be. Once the player has made the choice, it then moves the piece to the desired position and orientation. Afterwards, another piece will be spawned and the player has to repeat the same process. This will continue until the next piece can no longer be spawned, which means that the top row of the Tetris board is already occupied, or the maximum number of spawned pieces has been reached. In a Standard Tetris Game, only the current piece is known. Some other versions provide a preview of the next piece.

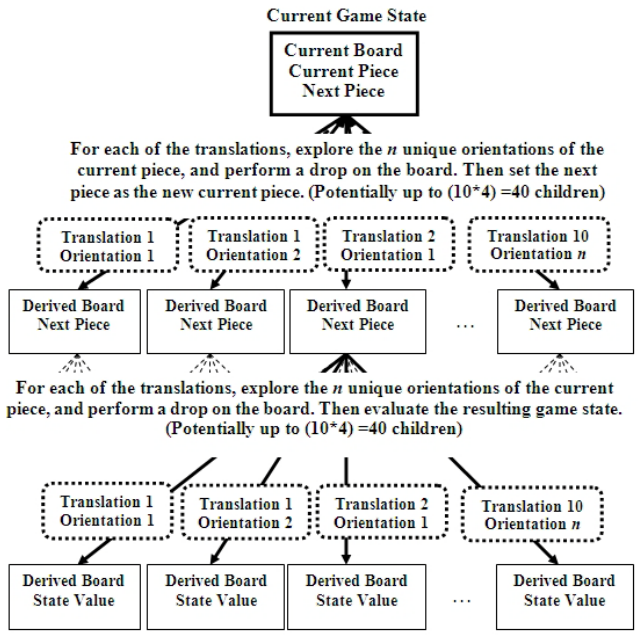
Current Game State
Current Board
Current Piece
Next Piece
For each of the translations, explore the n unique orientations of the current piece, and perform a drop on the board. Then set the next piece as the new current piece. (Potentially up to (10*4) =40 children)
Translation 1 Orientation 1
Translation 1 Orientation 2
Translation 2 Orientation 1
Translation 10 Orientation n
Derived Board Next Piece
...
For each of the translations, explore the n unique orientations of the current piece, and perform a drop on the board. Then evaluate the resulting game state. (Potentially up to (10*4) =40 children)
Derived Board State Value

Fig. 14 The Tetris Agent Two-Piece Decision-making Process

In the case of knowing in advance the next piece, known as a *two-piece strategy*, the player may consider it in deciding for the next move. In this case, the player has to enumerate all the possible combinations of translations and orientations of the next piece for each of the derived game state of the current piece's possible moves. Then after that, the board is evaluated based on the preferences of the player. In this case, the state evaluation function is used. To ensure that the game will not end early, the player must select the move that has the highest state value. In this scenario, the player is performing a *greedy strategy*, in which the player always selects the move with the highest reward (state value). Figure 14 provides a graphical view of the decision-making process.

5. Experiment Results and Discussion

The goal of the experiments is to determine the efficiency of the Harmony Search algorithm as the underlying optimization algorithm for the Tetris intelligent agent and to test the ability of the intelligent agent, *Harmonetris*, in finding the best possible solution with respect to the spawned game pieces. Each setup was subjected to 30 runs to make sure that the results are statistically acceptable. In this experimental setup, the following parameters are defined:

- Memory Improvisation (Number of Cycles) = 100
- Harmony Memory Size (Musical Pieces) = 5
- Harmony Consideration / Acceptance Rate = 0.95
- Pitch Adjustment Rate = 0.99

The results of our first experimental setup determined the performance of Harmony Search algorithm as the underlying optimization algorithm. After executing 120 runs in all, it has been observed that the maximum number of rows that the Tetris agent can clear is determined by Eq. (2).

$$\max rows = (\#\ of\ spawned\ pieces\ /\ 2.5) - 1 \qquad (2)$$

According to C. Fahey [6], the theoretical best case for a Tetris game is to be able to clear one row using 2.5 numbers of spawned pieces. Such special case is illustrated in Figure 15 wherein a sequence of five "O" Tetrominos is able to clear two rows, thus the computed ratio of 5(spawned pieces) / 2(cleared rows) = 2.5.

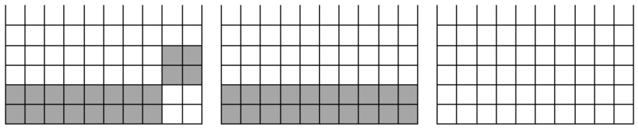

Fig. 15 The Optimal Tetris Case

This theory is taken from the fact that the Tetris board is 10 cells wide and a Tetromino has 4 cells each. This case however, is only applicable in instances wherein the number of spawned pieces is not random, which violates one of the basic specifications of Tetris.

Figure 16 to Figure 19 show the maximum number of cleared rows with respect to number of cycles over 30 runs with 100, 300, 500 and 1,000 spawned pieces, respectively.

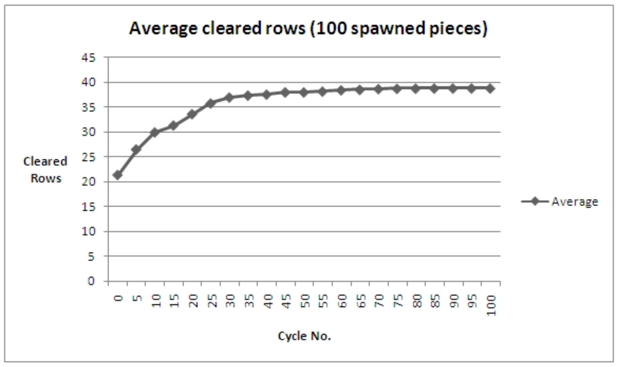

Fig. 16 Maximum number of cleared rows for 100 spawned pieces

The figures show the results of the experiments averaged over 30 runs. It is the average of the maximum number of cleared rows obtained by the best harmonies on all runs for a given cycle.

It can be observed from Figure 16 that the number of cleared rows approaches but does not reach 40. Thus, the maximum number of cleared rows in 30 runs for the 100 spawned pieces is 39.

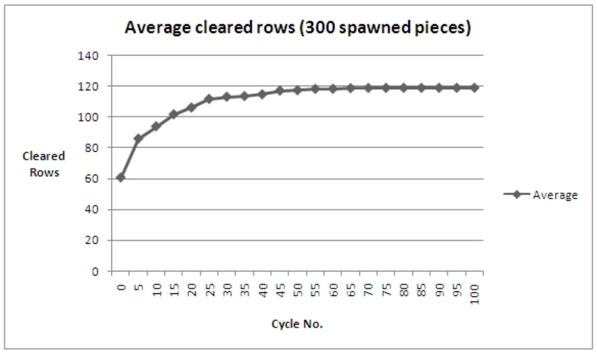

Fig. 17 Maximum number of cleared rows for 300 spawned pieces

Figures 17 to 19 also confirm the validity of Eq. (2), that is, the maximum number of rows that the Tetris agent can clear is determined by the said equation. Table 3 summarizes the experiment results obtained for the different setups.

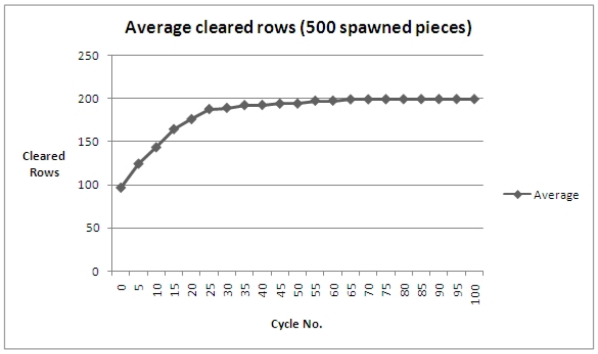

Fig. 18 Maximum number of cleared rows for 500 spawned pieces

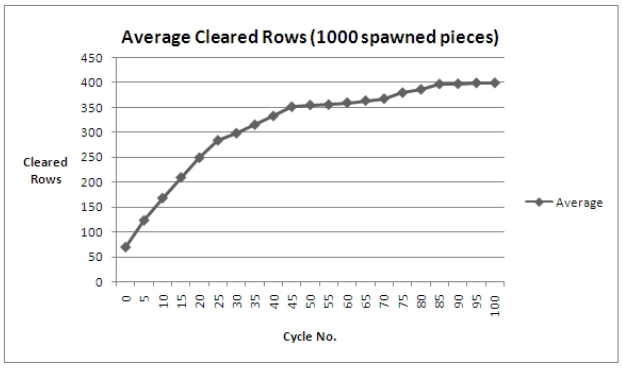

Fig.19 Maximum number of cleared rows for 1,000 spawned pieces

Table 3: Spawned Pieces and Maximum number of Cleared Rows

Total Number of Spawned Pieces (SP)	*Maximum number of Cleared Rows (CR)*	*Eq. (2) (SP/2.5) - 1*	*SP to CR Ratio*
100	39	39	2.56410
300	119	119	2.52100
500	199	199	2.51256
1,000	399	399	2.50626

Another experiment was conducted to determine the best configuration for the feature functions, thus the terminating condition was set to “Game Over” only, with no restriction on the number of cycles and spawned pieces.

After allowing the program to run for two weeks straight, the Tetris agent was able to obtain the following harmonies, x_i, on its 304th cycle of harmony improvisation. Table 4 shows the results obtained by the 5 harmonies on the 304th cycle.

Table 4: The Performance of the Five Harmonies on the 304th cycle

Harmony	*Maximum number of Cleared Rows (CR)*	*Total Number of Spawned Pieces (SP)*	*SP to CR Ratio*
x_1	291,087	727,751	2.500115085867
x_2	300,277	750,723	2.50010157288
x_3	337,254	843,168	2.500097849098
x_4	348,047	870,151	2.50009625136
x_5	416,928	1,042,354	2.50008154885

As shown in Table 4, it can be observed that as solutions improve resulting to more cleared rows, the ratio of spawned pieces (SP) to cleared rows (CR) approaches the value of our theoretical best game play of 2.5.

Furthermore, analysis on the harmonies (weight configurations) reveals that the Tetris agent, in its attempt to come up with a better solution, gave emphasis on reducing the weight values of the number of holes, wells, column transitions and row transitions, and at the same time increasing the weight value of potential rows.

6. Conclusions

In this paper, the researchers showed the efficiency of the Harmony Search algorithm as the underlying optimization algorithm for the Tetris intelligent agent and tested the ability of the intelligent agent in finding the best possible solution with respect to the random sequence of spawned game pieces. Experiment results reveal that the Harmony Search algorithm is an efficient optimization algorithm and the *Harmonetris*, the Tetris agent, is able to generate the best possible solution.

The best harmony (weight configuration) found in a span of two weeks was able to clear 416,928 rows in 1,042,354 spawned pieces yielding the Spawned Pieces to Cleared Rows ratio (SP/CR ratio) of 2.50008154885256. Thus, it can be observed that as the number of cleared rows increases, the SP/CR ratio approaches the optimum value of 2.5.

References

[1] A. Boumaza, “On the Evolution of Artificial Tetris Players”, In Proceedings of the 5th International Conference on Computational Intelligence and Games, 2009, pp. 381-393.
[2] N. Böhm, G. Kòkai and S. Mandl, “An Evolutionary Approach to Tetris”, MIC 2005: The Sixth Metaheuristic International Conference, 2005.
[3] X. Chen, H. Wang, W. Wang, Y. Shi and Y. Gao, “Apply Ant Colony Optimization for Tetris”, In Proceeding of the 2009 Genetic and Evolutionary Computation Conference, 2009.
[4] P. Collet and J.P. Rennard, Stochastic Optimization Algorithms, Handbook of Research on Nature Inspired Computing for Economics and Management, Hershey: IGR, 2007.
[5] E.D. Demaine, S. Hohenberger, and D. Liben-Nowell, “Tetris is Hard Even to Approximate”, In Proceedings of the 9th Annual International Conference on Computing and Combinatorics, Lecture Notes In Computer Science, 2003, pp. 351-363.
[6] C.P. Fahey, Tetris, http://www.colinfahey.com/tetris/tetris en.html.
[7] Z.W. Geem, J.H. Jim and G.V. Loganathan, “A new heuristic optimization algorithm: Harmony Search”, Simulation, 2001, 76(2), pp. 60-68.
[8] X-Z. Gao, X. Wang and S.J. Ovaska, “Uni-modal and Multi-modal Optimization Using Modified Harmony Search Methods”, International Journal of Innovative Computing, Information and Control, Volume 5, Number 10(A), 2009, pp. 2985-2996.
[9] Z. Kong, L. Gao, L. Wang, Y. Ge and S. Li, “On an Adaptive Harmony Search Algorithm”, International Journal of Innovative Computing, Information and Control, Volume 5, Number 9, 2009, pp. 2551-2560.
[10] Z.W. Geem, “State-of-the-Art in the Structure of Harmony Search Algorithm”, Studies in Computational Intelligence, vol. 270, 2010 pp. 1-10.
[11] Z.W. Geem, M. Fesanghary, J-Y. Choi, M.P. Saka, J.C. Williams, M.T. Ayvaz, L. Li, S. Ryu and A. Vasebi, Recent Advances in Harmony Search (Studies in Computational Intelligence), 1st edition, 2010.
[12] K.S. Lee and Z.W. Geem, “A New Meta-Heuristic Algorithm for Continuous Engineering Optimization: Harmony Search Theory and Practice”, Computer Methods in Applied Mechanics and Engineering, 194(36-38), 2005, pp. 3902-3933.
[13] S-B. Rhee, K-H. Kim and C-H. Kim, “Harmony Search Algorithm for Emission Considering Economic Load Dispatch Problems in Power Systems”, International Journal of Innovative Computing, Information and Control-Express Letters, vol. 3, Issue 4(B), 2009, pp. 1269-1274.
[14] X-S. Yang, “Harmony Search as a Metaheuristic Algorithm”, Music-Inspired harmony Search Algorithm-Theory and Applications, vol. 191, 2009, pp. 1-14.