CS 4701 Artificial Intelligence Practicum

Web-based Game
Unbeatable AI Tetris

Team members:
Xi He (netID: xh243)
Yanjia Li (netID: yl2493)

Description

This project is a web-based game design using Artificial Intelligence techniques. The application finally completed is a reactive and rule-based system that knows how to play tetris by itself and relatively optimal. The game supports two modes: AI show, which AI plays to show how much it can score, and game playing competition, which the user plays tetris game to compete against the computer agent. Refer to section of Game Explanation for detail of how to play the designed game. With great methods used and implementation, the system is finally intelligent enough to be able to beat human tetris players and is even able to score thousand times more than human tetris players through the proposed AI algorithm.

Method Implementation

Overview

In Tetris playing game[5], score is calculated as number of rows removed in the game. To achieve more number of rows removed, the agent needs to consider where to put the random shape given by the computer and what the orientation should be at each play. The agent needs to think reasonably from different aspects and some of these aspects may weigh more whereas some may weigh less(just like human playing tetris). Therefore the approach to implement this system is first defining 6 features describing resulting consequence of possible tetris moves (i.e tetris placement including it's orientation and location) and then construct an evaluation function for evaluating these tetris possible moves with these 6 feature. That is, giving an input state, the fitness evaluation function will calculate the evaluation score of each possible move and return the agent the move with the best evaluation score (not the game score). In this algorithm, initially the evaluation function with the 6 features is defined to inform the agent which aspects should be considered and evaluated before picking a move / making a decision. Then Particle Swarm Optimization is applied to optimize the function by adjusting the weights of the 6 features to achieve the best performance of the computer agent (make smarter moves to removes rows more overall). These weights helps agent make relatively optimal moves by telling the agent which features should be considered more which should be less. To be specific, the procedures are explained below:

Fitness function with features

The 6 features are very reasonable in evaluation which are summarized and referred from the existent state-of-art Tetris playing AI methods. The evaluated score f(move) is defined as the function below:

$f(move) = w1 \times f1 + w2 \times f2 + w3 \times f3 + w4 \times f4 + w5 \times f4 + w6 \times f6$, where

<u>f1 is landing height:</u>

The height of the highest block in the tetris game space after the move

f2 is the rows will be removed:

Rows will be removed by taking the move

f3 is row transitions:

Counts number of times each row transits from the left to the right then add all of them up. For example: 101111 would be 2 row transitions which are at first column and second column. It goes from 1 to 0 which counts as one, then it goes from 0 to 1 which counts another one. Doing the same thing to all rows then add all of them up to get to total row transition in the current state/tetris game space after the move is taken.

f4 is column transitions:

Same concept as f3 but applied on each column instead.

f5 is number of holes:

Counts the number of holes in each column then add them up to get the total number of holes. For each column, it searches from the top block down to the bottom edge in the tetris play space. Whenever meets an occupied block, it counts the single or continuous empty blocks connected right below the occupied block. This would get one column holes then sum up all the holes from each column.

f6 is well sums:

Counts the number of wells in a column that is empty but the left and right positions are occupied with blocks (walls are considered with blocks occupied for counting purpose). Then all the columns add together. (Note: for each column, it searches down from the top block to the bottom edge one by one. Whenever it meets a well block(say A), it finds the single or all the continuous well blocks connected right below it. Then it keeps searching down from the block right below the well block A . This means the wells in the column might be counted multiple times. i.e the lower, the more times which intuitively emphasis that the lower well is more risky and hard to handle)

Particle Swarm Optimization

PSO[4] is applied to adjust these weights to optimize the performance of the agent such that the agent can score (rows removed) relatively optimal to beat human players and can even removes rows thousand times more than human players. Since there are 6 feature, there are 6 dimensions in the search space. In this algorithm, a range of -100 to 100 for each dimension is used for optimal weights searching. 63 players / particles are specified to play the game each round. The starting position and the velocity of these players in the search space are randomly generated at the very beginning. After the first round play, best global score (rows removed) of these players are found and each personal best score is found as well (it's itself in the first round). Then the velocity of each player is updated using the PSO function below:

The velocity of particle m at dimension i is then

$$p[m].v[i] =$$
$$para1 \times p[m].v[i] +$$
$$para2 \times (p[m].pBestWeight[i] - p[m].weight[i]) \times Rand +$$
$$para3 \times (gBestWeight[i] - p[m].weight[i]) \times Rand$$

This is standard PSO function, where
p[m] is particle m
v[i] is particle m's velocity at dimension i
pBestWeight is the personal best weight of particle m at dimension i.
gBestWeight is the global best weight of these particles at dimension i.
Rand is a random value generated.
Para1,2,3 are the pre-defined parameters used to adjust the changing rate which are 0.6571, 1.6319, 0.6239 respectively. These values are suggested to achieve a good optimization performance.

Once the velocity is updated, the position of each particle is updated then by adding the velocity with the position location (weights) for each dimension. These players keep playing with the updated weights to find the new score then do the same thing. This entire procedure is executed over and over again to eventually find the global best that would make the agent score the most. Ideally, the position(weights) of each particle should converge but require a large amount of time and rounds. In this project, PSO executed the game for 75 times for each particle which takes about 8 hours in total. The global best weights stop changing in round 40. These particles doesn't eventually coverage (may converge if running hundreds of more times) but the relative optimal best score is a found at round 40 which is already able to make the agent to achieve a rows removed of 24004 in the optimization process shown in the optimization process log (only the 75th round) in Figure 1 in the next page. It shoulds all the 63 player's result and shows the global best score in the total 75 rounds.

The best weights found within the 75 rounds are:

w1 = -60.017980136357515
w2 = 39.38972760181724
w3 = -31.934414083175437
w4 = -90.04110337416141
w5 = -87.65903423606353
w6 = -64.13380030664379

```
-->  ********************************************************************************************************************
-->
-->
--> rows removed: 0 for weights: 48.14748852583106, 63.97321454585135, 30.438765185408585, 9.030195518190894, 81.93945511838623, 99.76872562861762
--> rows removed: 1755 for weights: -62.43706918274265, 36.511599219370794, -29.388861308802824, -92.84683302998006, -91.123676145089, -64.74737725595399
--> rows removed: 1284 for weights: -71.47249735709045, 33.4156650505647, -30.909706316130986, -96.51098952964674, -98.77613089102847, -78.5525364465522
--> rows removed: 1944 for weights: -64.83288039426206, 40.679915005169995, -33.17280133397864, -89.95598876412096, -88.23148573336026, -64.0249380082816
--> rows removed: 530 for weights: -59.93986841064998, 35.764764909505104, -33.89617597632204, -95.28197653476565, -88.30927845202167, -64.11983035734306
--> rows removed: 1556 for weights: -59.89111304284839, 38.51088553789416, -31.254595639995486, -91.10111050981122, -91.39244348896244, -58.457580282947404
--> rows removed: 923 for weights: -72.06923576070893, 32.83732281674755, -31.557641802157566, -96.66238934681162, -94.35540883678009, -60.11170761456023
--> rows removed: 3627 for weights: -66.0718077523364, 41.445313488531625, -35.03951022072293, -72.31878116874066, -86.97882224642429, -66.41152934867074
--> rows removed: 1933 for weights: -66.48158325996506, 27.57504562318192, -32.22751516143805, -93.22771948380736, -80.86832532658732, -64.45870517551623
--> rows removed: 1368 for weights: -61.564824016244565, 25.81255538678121, -26.55524117510724, -90.3277483104054, -90.7961103791881, -67.57642245801082
--> rows removed: 2835 for weights: -59.99377427529654, 39.389800129971215, -31.943910175808572, -90.03895481529355, -87.6495565091567, -64.13700284757901
--> rows removed: 306 for weights: -60.09483101189756, 36.27118803242773, -31.98094133412019, -81.21767400721922, -95.2889895864813, -64.69599344397311
--> rows removed: 2170 for weights: -79.16035985796307, 39.06995417983284, -33.726207612005915, -90.34073367132817, -86.29734491592255, -69.14180637149599
--> rows removed: 709 for weights: -78.8439461495598, -10.77496037315653, -35.50399253199798, -89.00641750968474, -89.18167278788721, -61.52774036034555
--> rows removed: 2318 for weights: -67.56692493173372, 47.91840500579394, -34.562308912033444, -99.49450604336866, -89.99824344912285, -90.88158689754005
--> rows removed: 2721 for weights: -74.14229931842036, 40.977250284913936, -43.40822666112129, -93.03148063722496, -90.1874762363484, -73.62601090447049
--> rows removed: 3497 for weights: -62.7760093059197, 38.41353851274087, -39.669008976521715, -99.41005669262834, -89.30265202313143, -57.591933031834046
--> rows removed: 3834 for weights: -67.63284175049802, 30.343846370800964, -30.363206361016452, -99.4069115632707, -92.38948939706725, -67.18330895297841
--> rows removed: 1048 for weights: -59.9858300604028, 38.445546244613084, -32.63678041525267, -90.0878598284031, -94.68032316995009, -63.597644091951885
--> rows removed: 100 for weights: -72.41546761211816, 48.757068814336954, -22.38508239265357, -84.03853105862038, -88.27271258158619, -59.80222507471155
--> rows removed: 379 for weights: -60.18480168327039, 32.30208792574591, -30.797961885567517, -91.14807222488679, -89.70777357589175, -59.086641660977726
--> rows removed: 1446 for weights: -58.18200965071358, 37.68640883589032, -38.03257596237472, -96.779411890045, -86.42828830041542, -59.60856486050412
--> rows removed: 5440 for weights: -48.239012678933754, 52.656778575551969, -43.22297362818875, -89.68935204878262, -92.542525015414, -66.87881917182564
--> rows removed: 5427 for weights: -59.384898200610166, 36.720428530145966, -31.583011842585464, -90.5259602879329, -89.43947239873715, -62.215670404970375
--> rows removed: 2902 for weights: -59.74692739705793, 40.745136519288884, -34.160371509136546, -90.6382236423838, -91.76098706318069, -63.42676242400097
--> rows removed: 3155 for weights: -66.0826089733907, 59.2782467378664, -38.0925606051772, -95.56189170482358, -96.48588068084791, -52.69738286397657
--> rows removed: 6441 for weights: -59.47866397641436, 38.920352042792246, -35.359932200896345, -90.55685169871812, -90.588273498887993, -63.966901385088111
--> rows removed: 679 for weights: -62.451372733361524, 30.063422235331945, -33.37906013837556, -87.196679096813, -97.09701417962707, -90.26558176331014
--> rows removed: 12498 for weights: -66.32556803280625, 26.468330288099207, -33.138243795178404, -95.70956919046897, -94.40151205825707, -65.29359326835907
--> rows removed: 1169 for weights: -58.36811268551895, 21.3165283189904, -22.822506673798333, -94.33376335459623, -89.92440413491174, -60.71038470934825
--> rows removed: 403 for weights: -60.02140850576169, 40.845179987620355, -29.958675057142838, -90.73764855917942, -84.25991885460908, -63.907469211443974
--> rows removed: 708 for weights: -67.70887329591108, 40.281252632088275, -32.06876653786533, -93.93328074873901, -93.09131402716997, -63.483615723008846
--> rows removed: 2682 for weights: -71.92283593544288, 34.501632513460216, -29.892732384970433, -94.77824440533799, -88.45743437757284, -62.747440594003216
--> rows removed: 774 for weights: -66.45633220543584, 39.711283871622896, -31.830927810576, -88.907759789696, -89.99518266247291, -55.39470643566692
--> rows removed: 1901 for weights: -70.86998886224038, 41.82707535249318, -31.905162889900364, -87.5416575674541, -87.73042708402407, -64.10185677693191
--> rows removed: 4369 for weights: -61.462010078654, 39.27770288799882, -32.10489241419944, -92.04629494427174, -89.57405116759026, -64.0320246687033
--> rows removed: 6147 for weights: -64.40497256666751, 21.551216931242656, -32.85734852739897, -88.68556620631139, -88.52519307761092, -63.158129194974888
--> rows removed: 955 for weights: -60.14448646483988, 37.44431462392835, -34.03535150141419, -91.00182912129449, -96.35135926096301, -59.1021446534961
--> rows removed: 415 for weights: -66.32816228929568, 38.04561757656943, -36.68836665542467, -81.82663961957141, -91.5714853440405, -59.914960647899704
--> rows removed: 5235 for weights: -63.30018692297349, 29.542100588673836, -33.09760845541273, -90.06283121488958, -88.05711857573519, -64.82446418911923
--> rows removed: 313 for weights: -45.29653466096933, 59.51671047429819, -49.1245279314657735, -94.42082447106759, -92.62335326812101, -54.25821040076883
--> rows removed: 292 for weights: -79.7395761178856, 12.343556813149597, -55.87416246216367, -97.02577852218955, -96.36442442701187, -62.64996128702458
--> rows removed: 134 for weights: -59.62113729136325, 39.41727100024472, -31.378073170682107, -88.94689949371666, -81.64789720454297, -63.17090299649523
--> rows removed: 751 for weights: -63.80134330828528, 38.47621545107136, -39.3109596412761O6, -97.40180395618135, -90.98352014953086, -62.537364391457955
--> rows removed: 2075 for weights: -72.39262082867874, 40.987517983294836, -33.00020022145561, -96.6343873998515, -92.89216435780877, -74.84912754656041
--> rows removed: 3133 for weights: -65.04916351510276, 38.23172151251842, -34.38978820084264, -95.20708985371485, -90.70561039718969, -61.73119323654434
--> rows removed: 4314 for weights: -67.77049840958003, 28.867009623083476, -33.2545021324969674, -91.49666114928417, -91.9060076094506, -57.300387086265688
--> rows removed: 2709 for weights: -77.67825510734151, 32.48851472487644, -32.11444682780301, -92.07436124967897, -95.92686393643623, -60.34877270132477
--> rows removed: 2138 for weights: -89.61530064168309, 35.96955462075265, -25.870537917393058, -95.49234015969238, -91.66771822572288, -65.46260235091589
--> rows removed: 2654 for weights: -87.21117011781101, 39.18314056859147, -22.55855332666049, -84.32785508534384, -96.02296627684684, -47.482516055283526
--> rows removed: 6181 for weights: -61.13300347388995, 29.231831818521908, -30.218032344591457, -91.25349840061448, -90.50859999234329, -58.470846214142824
--> rows removed: 891 for weights: -54.55877135702021, 9.875696107859355, -34.193347564709896, -89.93495611663057, -87.96112448998687, -60.516612033499406
--> rows removed: 2905 for weights: -62.702325588185964, 35.70927469084981, -32.50089992789094, -91.9874479684149, -90.735475769775536, -65.23365193793875
--> rows removed: 331 for weights: -58.58028294040609, 39.457128913336824, -31.646653765268887, -91.57432308472808, -87.68254035438407, -63.72805485087764
--> rows removed: 393 for weights: -59.27500378268112, 40.544982174351546, -35.22783339917245, -94.36519649395485, -86.96349672504391, -64.07272943278713
--> rows removed: 3733 for weights: -66.48891609557285, 29.252633460849903, -37.79039173920248, -93.41254869189227, -89.0022474687871, -77.67941176079415
--> rows removed: 2476 for weights: -58.841907115788224, 39.10672561948953, -30.407682473945872, -92.09451141857963, -86.01691166735425, -61.94796138410492
--> rows removed: 7622 for weights: -63.622589726864646, 46.97312990383903, -32.72802458975237, -92.2493642299837, -96.20761036070613, -76.93810610271264
--> rows removed: 783 for weights: -60.32371455402864, 20.513118243766083, -40.3200427496989, -90.2012512579742, -90.11041176343173, -55.335304114878475
--> rows removed: 2620 for weights: -60.31149348677383, 39.3397714088394, -19.77587000284283, -85.08546127976265, -89.05408584592438, -58.576286790647764
--> rows removed: 6135 for weights: -60.56721420316762, 34.945656211153675, -33.633610442899005, -90.45003059111568, -87.16975329929241, -68.1115657788488
--> rows removed: 1227 for weights: -81.08616386554752, 39.252343620113486, -19.971992683269715, -90.88748911673143, -91.86455228545381, -62.938557038864966
--> rows removed: 447 for weights: -60.422402020174175, 35.67460634177888, -37.458414369426606, -94.99164479964085, -76.18927260925133, -66.12118199486005
-->
-->  ********************************************************************************************************************
-->
-->  Best Score in iter 75: 24004 with --> -60.017980136357515,39.38972760181724,-31.934414083175437,-90.04110337416141,-87.65903423606353,-64.13380030664379
-->
-->  ********************************************************************************************************************
```

Figure 1. The 75th round result

63 players result in 75th round and global best weights of all 75 rounds shown in the last line

These weights look very reasonable since it tells which feature is more important which is less and can be understood intuitively by human.

This then constructs our evaluation function below:

```
function evaluateDecision(testResult,orientation) {

      return GetLandingHeight(testResult, orientation) * -60.017980136357515 +
          testResult.rows_removed * 39.38972760181724+
          GetRowTransitions(testResult.map_temp) * -31.934414083175437 +
          GetColumnTransitions(testResult.map_temp) * -90.04110337416141+
          GetNumberOfHoles(testResult.map_temp) * -87.65903423606353 +
          GetWellSums(testResult.map_temp) * -64.13380030664379;
}
```

Testing Implementation and Result

Using the developed algorithm/application, a series of tests are conducted to check the performance of the evaluation function with its weights. The graph below in Figure 2 is the testing result of running AI self game playing for 30 times. As we can see, the best round which final number of rows removed in the game is 59760 shown in the Figure 3 below. The score is fluctuating up and down but all of them can achieve at least above 20000 with one exception which only achieves 229. Since computer generates block shape randomly, the random factor can affect the performance as well. i.e. maybe a sequence of bad blocks are generated which decreases the agent's performance. The agent plays very fast if it's in mode of AI show. It makes 100 moves in one second meaning a lot of CPU is used which may somewhat affect the randomness of the shapes. As we also know, the randomness is not truly random in a computer. In conclusion, the overall performance is very excellent because it achieves only 24004 in the optimization process whereas it achieves way higher than that at the testing evaluation process.

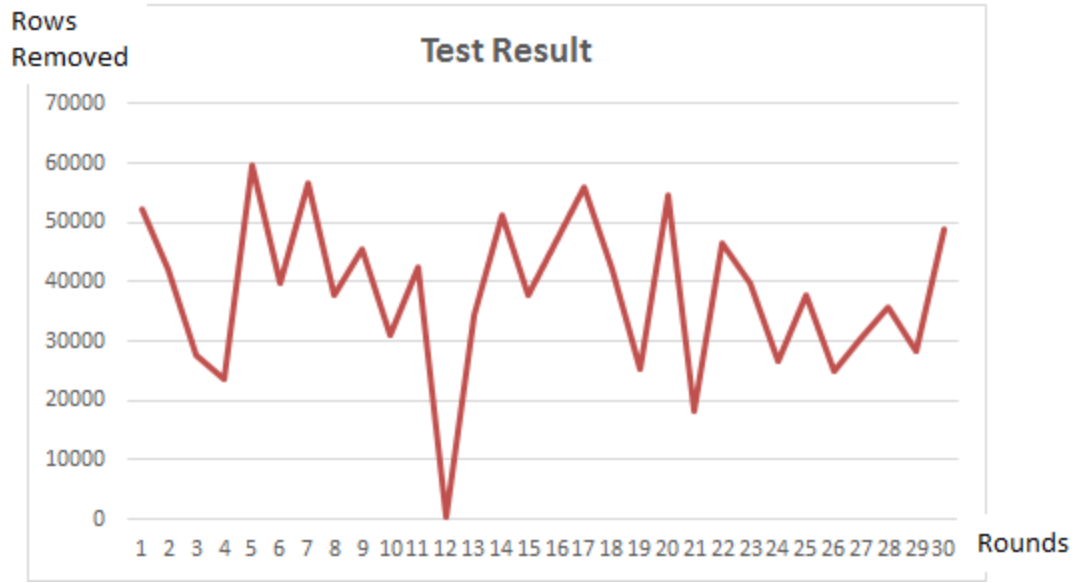

Figure 2. Rows removed result from 30 times test running

Figure 3. Best Result of the Tests

Game Explanation

Web-based unbeatable AI tetris is an online game in which you can play with our elaborately designed AI. Player and AI will put a random shape into their playground in turns. The main goal of this game is to eliminated as many as possible lines before the game finish. The game will finish either when AI or player has too many lines in his area.

Fig 4 shows the graphical interface of our game and we will talk about more details next. As we can see from the figure, there are mainly four section. Two rectangle area is the playground for AI and player. Their blocks will drop into this area. There is also a console area between two playgrounds. In the console area, the number of lines which AI or player eliminated, the number of level, next incoming shape and time left for next move will be given. At the left -most is the setting area. You can press button “instruction” to see the basic control for the game. Press “Challenge” to start a game together with AI. The drop-down menu “Difficulty” contains different challenge modes for player. We will talk about it later. Button “AI show” is used to show how AI plays tetris alone. Therefore, you can have a quick view of how AI works. To change the speed of automatically AI show, you can use the drop-menu “Show speed”. What’s more, if you run into some problem while playing, you can click the button “reload” to start a completely new game. Be careful, all the work you have done before will be

lost. Finally, for people who do not have a real keyboard, we provide a simple set of virtual arrow key. Although it is hard to control by mouse or touch, it helps a little.

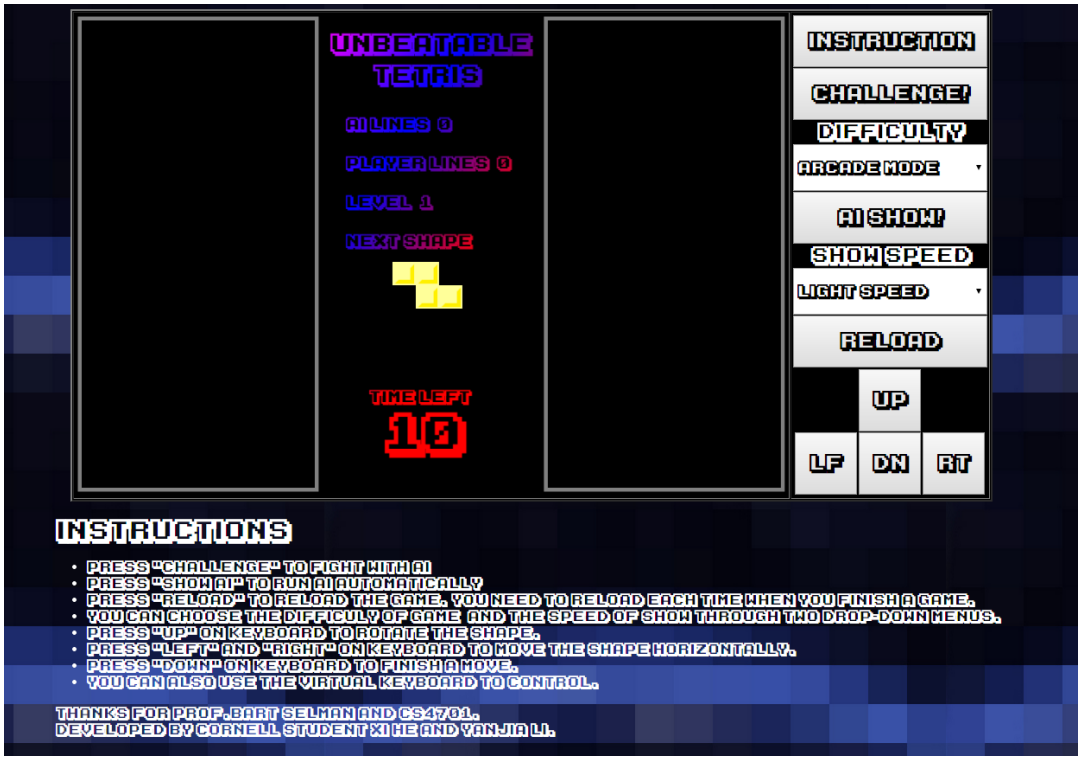

Figure 4 Graphical interface of web-based unbeatable AI tetris

Next we will explain how to play with AI more specifically. We use arrow keys to control the coming shape. Press “Down” will make the shape drop. Press “Up” will change the orientation of the shape. “Left” and “Right” is used to change the horizontal position.

When you choose to challenge our AI, you should select a mode from five pre-defined modes first. They are “Arcade mode”, “Extremely hard”, “Hard”, “Easy” and “Free mode”. The latter four modes are just for practicing because you can just consider about the strategy of where to put the shape. You don’t need to worry about other troubles. The difference between “Extremely Hard”, “Hard”, “Easy” and “Free mode” is only how much time do you have to finish a move. For example, in “Free mode”, you have more than 900 seconds to decide a move so you can have enough time to think out a best play. There is another special mode called “Arcade mode”, in this mode, you need to try to get higher level before the game finish. In this mode, there are many other influential factors like stones. Some stones will be put into your landscape once AI eliminate enough lines. And you can also put some stones to interfere AI by eliminating enough lines.

The entire algorithm is implemented using JavaScript including the optimization. The user interface and graphics is implemented using HTML, CSS. It is also implemented without any

framework. Although this leads to more complexity, it is a better way to fully understand how the game, AI algorithm and graphical interface works together. In this design, the playground is a 20 rows and 10 columns rectangle area. This area is implemented by a two dimensional array. Inside each element of the array is the RGB value of the color of that block. The empty block will have color “#000000”. Therefore, we can distinguish empty blocks from not empty ones. To find out how many lines eliminated by AI or player, we check each row from top to down. If the entire row is not empty, we move every block down.

Conclusion

This project is a web-based game design using multiple Artificial Intelligence algorithms. The application is intended to be a reactive and rule-based system that knows how to play tetris by itself and relatively optimal. The game supports two modes: AI show which AI plays to show how much it can score and game playing competition which the user plays tetris to compete the computer agent. After optimization through hundreds of plays, the system is eventually intelligent enough to be able to beat human tetris players and is even able to score (number of rows removed) thousand times more than human tetris players. To sum up, this design combined artificial intelligence algorithm with game design techniques, which turns out to be a good example for us to understand artificial intelligence further.

Reference

[1] Unknown. Applying Artificial Intelligence to Nintendo Tetris[EB/OL]. [2014-11-1]. http://meatfighter.com/nintendotetrisai/.

[2] Islam. El-Tetris: An Improvement on Pierre Dellacherie's Algorithm[EB/OL]. [2014-11-15]. http://ielashi.com/el-tetris-an-improvement-on-pierre-dellacheries-algorithm/.

[3] Russell S, Norvig P. Artificial Intelligence: A Modern Approach[M]. Prentice Hall, 2009.

[4] Wikipedia. Particle swarm optimization[EB/OL]. [2014-12-1]. https://en.wikipedia.org/wiki/Particle_swarm_optimization.

[5] Wikipedia. Tetris[EB/OL]. [2014-11-1]. https://en.wikipedia.org/wiki/Tetris.