





Using Artificial Intelligence to solve the 2048 Game (JAVA code)

(http://blog.datumbox.com/using-artificial-intelligence-to-solve-the-2048-game-java-code/)

 April 7, 2014  Vasilis Vryniotis (http://blog.datumbox.com/author/bbriniotis/)

 6 Comments (http://blog.datumbox.com/using-artificial-intelligence-to-solve-the-2048-game-java-code/#comments)

 Machine Learning & Statistics (/topics/machine-learning-statistics) Programming (/topics/programming)

634





1623



329



Share

179



Share

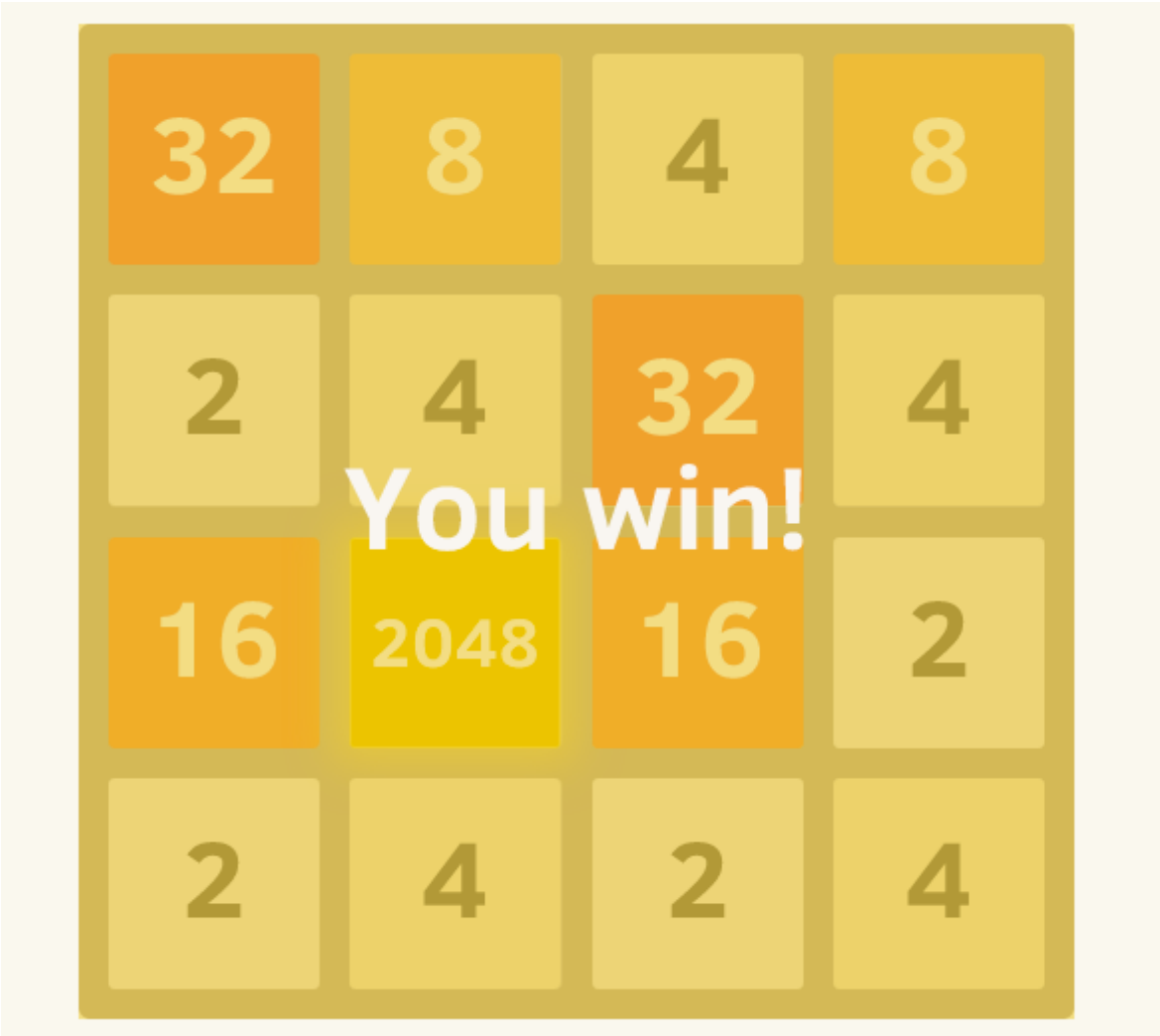
2811



Share

22





By now most of you have heard/played the [2048 game \(http://gabrielecirulli.github.io/2048/\)](http://gabrielecirulli.github.io/2048/) by Gabriele Cirulli. It's a simple but highly addictive board game which requires you to combine the numbers of the cells in order to reach the number 2048. As expected the difficulty of the game increases as more cells are filled with high values. Personally even though I spent a fair amount of time playing the game, I was never able to reach 2048. So the natural thing to do is to try to develop an AI solver in JAVA to beat the 2048 game. 😊

In this article I will briefly discuss my approach for building the Artificial Intelligence Solver of Game 2048, I will describe the heuristics that I used and I will provide the complete code which is written in JAVA. The code is open-sourced under GPL v3 license and you can download it from [Github \(https://github.com/datumbox/Game-2048-AI-Solver\)](https://github.com/datumbox/Game-2048-AI-Solver).

Developing the 2048 Game in JAVA

The original game is written in JavaScript, so I had to rewrite it in JAVA from scratch. The main idea of the game is that you have a 4x4 grid with Integer values, all of which are powers of 2. Zero valued cells are considered empty. At every point during the game you are able to move the values towards 4 directions Up, Down, Right or Left. When you perform a move all the values of the grid move towards that direction and they stop either when they reach the borders of the grid or when they reach another cell with non-zero value. If that previous cell has the same value, the two cells are merged into one cell with double value. At the end of every move a random value is added in the board in one of the empty cells and its value is either 2 with 0.9 probability or 4 with 0.1 probability. The game ends when the player manages to create a cell with value 2048 (win) or when there are no other moves to make (lose).

In the original implementation of the game, the move-merge algorithm is a bit complicated because it takes into account all the directions. A nice simplification of the algorithm can be performed if we fix the direction towards which we can combine the pieces and rotate the board accordingly to perform the move. [Maurits van der Schee \(http://www.leaseweblabs.com/2014/03/text-mode-2048-game-c-algorithm-explained/\)](http://www.leaseweblabs.com/2014/03/text-mode-2048-game-c-algorithm-explained/) has recently written an article about it which I believe is worth checking out.

All the classes are documented with Javadoc comments. Below we provide a high level description of the architecture of the implementation:

1. Board Class

The board class contains the main code of the game, which is responsible for moving the pieces, calculating the score, validating if the game is terminated etc.

2. ActionStatus and Direction Enum

The ActionStatus and the Direction are 2 essential enums which store the outcome of a move and its direction accordingly.

3. ConsoleGame Class

The ConsoleGame is the main class which allows us to play the game and test the accuracy of the AI Solver.

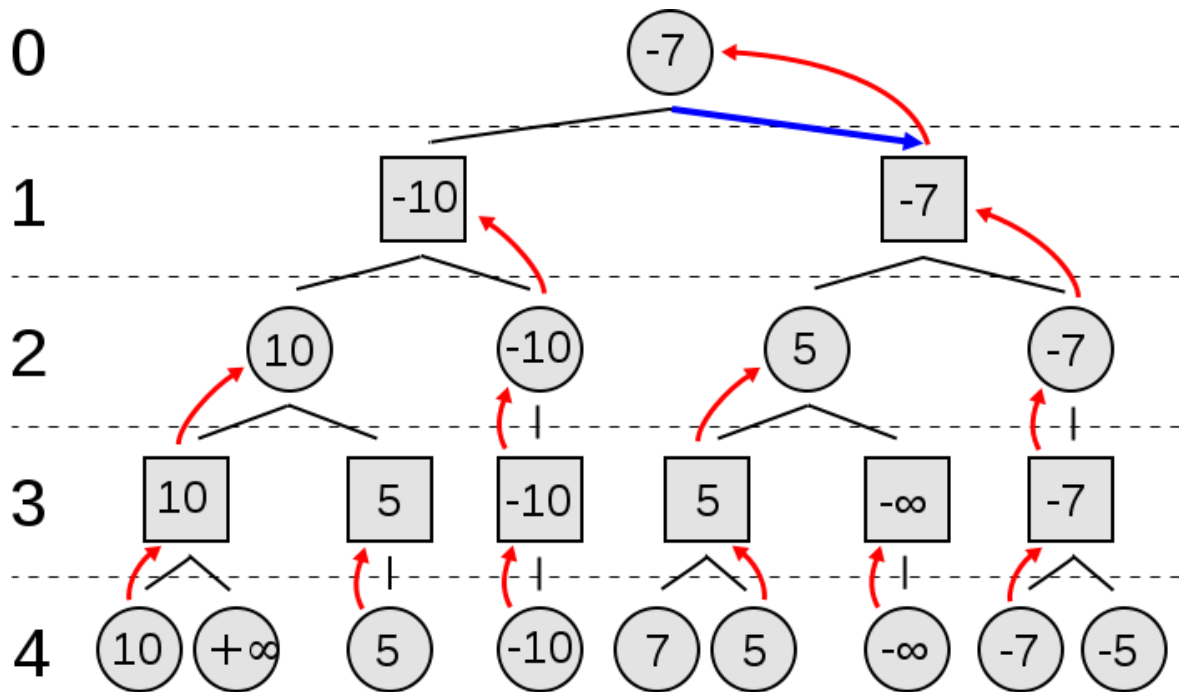
4. Alsolver Class

The Alsolver is the primary class of the Artificial Intelligence module which is responsible for evaluating the next best move given a particular Board.

Artificial Intelligence Techniques: Minimax vs Alpha-beta pruning

Several approaches have been published to solve automatically this game. The most notable is the one developed by [Matt Overlan](http://ov3y.github.io/2048-AI/) (<http://ov3y.github.io/2048-AI/>). To solve the problem I tried two different approaches, using Minimax algorithm and using Alpha-beta pruning.

Minimax Algorithm



The [Minimax](http://en.wikipedia.org/wiki/Minimax#Minimax_algorithm_with_alterate_moves) (http://en.wikipedia.org/wiki/Minimax#Minimax_algorithm_with_alterate_moves) is a recursive algorithm which can be used for solving two-player zero-sum games. In each state of the game we associate a value. The Minimax algorithm searches through the space of possible game states creating a tree which is expanded until it reaches a particular predefined depth. Once those leaf states are reached, their values are used to estimate the ones of the intermediate nodes.

The interesting idea of this algorithm is that each level represents the turn of one of the two players. In order to win each player must select the move that minimizes the opponent's maximum payoff. Here is a nice video presentation of the minimax algorithm:

Artificial Intelligence using Minimax



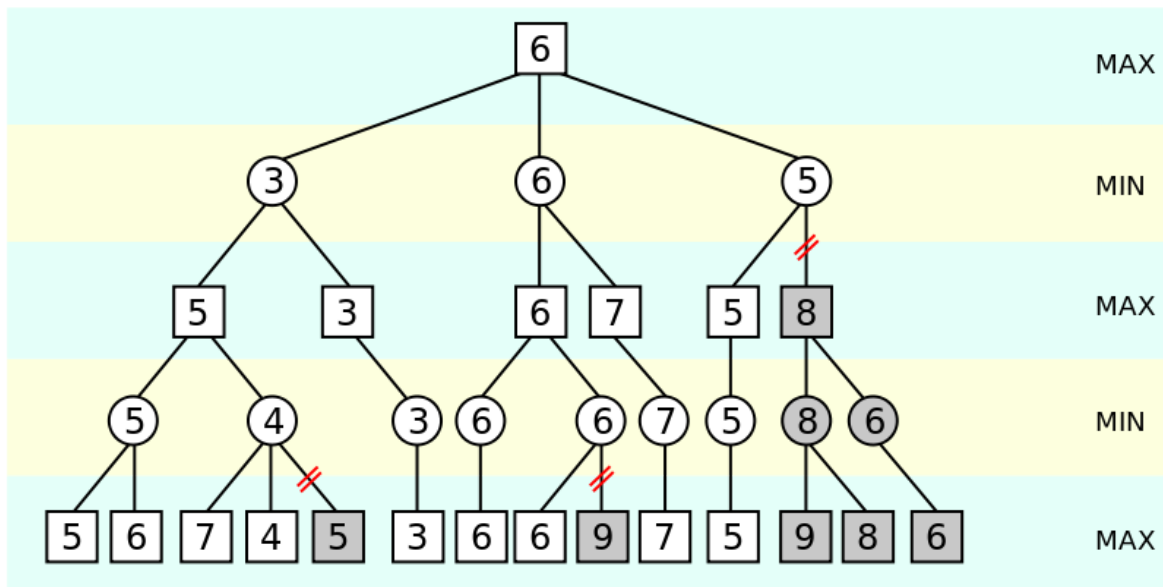
Below you can see pseudocode of the Minimax Algorithm:

```

function minimax(node, depth, maximizingPlayer)
  if depth = 0 or node is a terminal node
    return the heuristic value of node
  if maximizingPlayer
    bestValue := -∞
    for each child of node
      val := minimax(child, depth - 1, FALSE)
      bestValue := max(bestValue, val);
    return bestValue
  else
    bestValue := +∞
    for each child of node
      val := minimax(child, depth - 1, TRUE)
      bestValue := min(bestValue, val);
    return bestValue
(* Initial call for maximizing player *)
minimax(origin, depth, TRUE)

```

Alpha-beta pruning



The [Alpha-beta pruning algorithm](http://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning) (http://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning) is an expansion of minimax, which heavily decreases (prunes) the number of nodes that we must evaluate/expand. To achieve this, the algorithm estimates two values the alpha and the beta. If in a given node the beta is less than alpha then the rest of the subtrees can be pruned. Here is a nice video presentation of the alphabeta algorithm:

alpha-beta pruning



Below you can see the pseudocode of the Alpha-beta pruning Algorithm:

```

function alphabeta(node, depth,  $\alpha$ ,  $\beta$ , maximizingPlayer)
  if depth = 0 or node is a terminal node
    return the heuristic value of node
  if maximizingPlayer
    for each child of node
       $\alpha$  := max( $\alpha$ , alphabeta(child, depth - 1,  $\alpha$ ,  $\beta$ , FALSE))
      if  $\beta \leq \alpha$ 
        break (*  $\beta$  cut-off *)
    return  $\alpha$ 
  else
    for each child of node
       $\beta$  := min( $\beta$ , alphabeta(child, depth - 1,  $\alpha$ ,  $\beta$ , TRUE))
      if  $\beta \leq \alpha$ 
        break (*  $\alpha$  cut-off *)
    return  $\beta$ 
(* Initial call *)
alphabeta(origin, depth,  $-\infty$ ,  $+\infty$ , TRUE)

```

How AI is used to solve the Game 2048?

In order to use the above algorithms we must first identify the two players. The first player is the person who plays the game. The second player is the computer which randomly inserts values in the cells of the board. Obviously the first player tries to maximize his/her score and achieve the 2048 merge. On the other hand the computer in the original game is not specifically programmed to block the user by selecting the worst possible move for him but rather randomly inserts values on the empty cells.

So why we use AI techniques which solve zero-sum games and which specifically assume that both players select the best possible move for them? The answer is simple; despite the fact that it is only the first player who tries to maximize his/her score, the choices of the computer can block the progress and stop the user from completing the game. By modeling the behavior of the computer as an orthological non-random player we ensure that our choice will be a solid one independently from what the computer plays.

The second important part is to assign values to the states of the game. This problem is relatively simple because the game itself gives us a score. Unfortunately trying to maximize the score on its own is not a good approach. One reason for this is that the position of the values and the number of empty valued cells are very important to win the game. For example if we scatter the large values in remote cells, it would be really difficult for us to combine them. Additionally if we have no empty cells available, we risk losing the game at any minute.

For all the above reasons, several heuristics [have been suggested \(http://stackoverflow.com/a/22389702\)](http://stackoverflow.com/a/22389702) such as the Monotonicity, the smoothness and the Free Tiles of the board. The main idea is not to use the score alone to evaluate each game-state but instead construct a heuristic composite score that includes the aforementioned scores.

Finally we should note that even though I have developed an implementation of Minimax algorithm, the large number of possible states makes the algorithm very slow and thus pruning is necessary. As a result in the JAVA implementation I use the expansion of Alpha-beta pruning algorithm. Additionally unlike other implementations, I don't prune aggressively the choices of the computer using arbitrary rules but instead I take all of them into account in order to find the best possible move of the player.

Developing a heuristic score function

In order to beat the game, I tried several different heuristic functions. The one that I found most useful is the following:

```

1 | private static int heuristicScore(int actualScore, int numberOfEmptyCells, int clusteringScore) {
2 |     int score = (int) (actualScore+Math.log(actualScore)*numberOfEmptyCells -clusteringScore);
3 |     return Math.max(score, Math.min(actualScore, 1));
4 | }

```

The above function combines the actual score of the board, the number of empty cells/tiles and a metric called clustering score which we will discuss later. Let's see each component in more detail:

1. **Actual Score:** For obvious reasons, when we calculate the value of a board we have to take into account its score. Boards with higher scores are generally preferred in comparison to boards with lower scores.
2. **Number of Empty Cells:** As we mentioned earlier, keeping few empty cells is important to ensure that we will not lose the game in the next moves. Board states with more empty cells are generally preferred in comparison to others with fewer. A question rises concerning how would we value those empty cells? In my solution I weight them by the logarithm of the actual score. This has the following effect: The lower the score, the less important it is to have many empty cells (This is because at the beginning of the game combining the cells is fairly easy). The higher the score, the more important it is to ensure that we have empty cells in our game (This is because at the end of the game it is more probable to lose due to the lack of empty cells).
3. **Clustering Score:** We use the clustering score which measures how scattered the values of our board are. When cells with similar values are close they are easier to combine meaning it is harder to lose the game. In this case the clustering score has a low value. If the values of the board are scattered, then this score gets a very large value. This score is subtracted from the previous two scores and acts like a penalty that ensures that clustered boards will be preferred.

In the last line of the function we ensure that the score is non-negative. The score should be strictly positive if the score of the board is positive and zero only when the board of the score is zero. The max and min functions are constructed so that we get this effect.

Finally we should note that when the player reaches a terminal game state and no more moves are allowed, we don't use the above score to estimate the value of the state. If the game is won we assign the highest possible integer value, while if the game is lost we assign the lowest non negative value (0 or 1 with similar logic as in the previous paragraph).

More about the Clustering Score

As we said earlier the clustering score measures how much scattered are the values of the board and acts like a penalty. I constructed this score in such a way so that it incorporates tips/rules from users who "mastered" the game. The first suggested rule is that you try to keep the cells with similar values close in order to combine them easier. The second rule is that high valued cells should be close to each other and not appear in the middle of the board but rather on the sides or corners.

Let's see how the clustering score is estimated. For every cell of the board we estimate the sum of absolute differences from its neighbors (excluding the empty cells) and we take the average difference. The reason why we take the averages is to avoid counting more than once the effect of two neighbor cells. The total clustering score is the sum of all those averages.

The Clustering Score has the following attributes:

1. It gets high value when the values of the board are scattered and low value when cells with similar values are close to each other.
2. It does not overweigh the effect of two neighbor cells.
3. Cells in the margins or corners have fewer neighbors and thus lower scores. As a result when the high values are placed near the margins or corners they have smaller scores and thus the penalty is smaller.

The accuracy of the algorithm

As expected the accuracy (aka the percentage of games that are won) of the algorithm heavily depends on the search depth that we use. The higher the depth of the search, the higher the accuracy and the more time it requires to run. In my tests, a search with depth 3 lasts less than 0.05 seconds but gives 20% chance of winning, a depth of 5 lasts about a 1 second but gives 40% chance of winning and finally a depth of 7 lasts 27-28 seconds and gives about 70-80% chance of winning.

Future expansions

For those of you who are interested in improving the code here are few things that you can look into:

1. **Improve the Speed:** Improving the speed of the algorithm will allow you to use larger depth and thus get better accuracy.
2. **Create Graphics:** There is a good reason why Gabriele Cirulli's implementation became so famous. It is nice looking! I did not bother developing a GUI but I rather print the results on console which makes the game harder to follow and to play. Creating a nice GUI is a must.
3. **Tune Heuristics:** As I mentioned earlier, several users have suggested different heuristics. One can experiment with the way that the scores are calculated, the weights and the board characteristics that are taken into account. My approach of measuring the cluster score is supposed to combine other suggestions such as Monotonicity and Smoothness, but there is still room for improvement.
4. **Tuning the Depth:** One can also try to tune/adjust the depth of the search depending on the game state. Also you can use the Iterative deepening depth-first search (http://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search) algorithm which is known to improve the alpha-beta pruning algorithm.

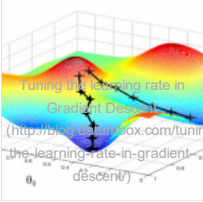
Don't forget to download the JAVA code from [Github \(https://github.com/datumbox/Game-2048-AI-Solver\)](https://github.com/datumbox/Game-2048-AI-Solver) and experiment. I hope you enjoyed this post! If you did please take a moment to share the article on Facebook and Twitter. 😊




About Vasilis Vryniotis (<http://blog.datumbox.com/author/bbriniotis/>)

My name is Vasilis Vryniotis. I'm a Data Scientist, a Software Engineer, author of Datumbox Machine Learning Framework and a proud geek. [Learn more](#) (<http://blog.datumbox.com/author/bbriniotis/>)


Related Posts:




Tuning the Learning Rate in Gradient Descent
(<http://blog.datumbox.com/tuning-the-learning-rate-in-gradient-descent/>)




Measuring the Social Media Popularity of Pages with DEA
(<http://blog.datumbox.com/measuring-social-media-popularity-with-dea/>)



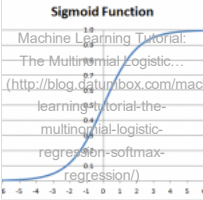
Efficiency
Quality
Data Envelopment Analysis
(<http://blog.datumbox.com/data-envelopment-analysis-tutorial/>)



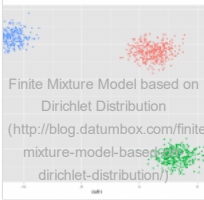
10 Tips for Sentiment Analysis Projects
(<http://blog.datumbox.com/10-tips-for-sentiment-analysis-projects/>)



Clustering with Dirichlet Process Mixture Model in Java
(<http://blog.datumbox.com/clustering-with-dirichlet-process-mixture-model-in-java/>)




Sigmoid Function
Machine Learning Tutorial: The Multinomial Logistic...
(<http://blog.datumbox.com/machine-learning-tutorial-the-multinomial-logistic-regression-softmax-regression/>)



Finite Mixture Model based on Dirichlet Distribution
(<http://blog.datumbox.com/finite-mixture-model-based-on-dirichlet-distribution/>)

Latest Comments



Vasilis Vryniotis (<http://blog.datumbox.com/author/bbriniotis/>)


Hi Athi,

Thanks for the comment. You are correct, this is indeed a bug.

What needs to be done is avoid passing a score from move, but actually return an object that gives more details on what happened inside. I will put it in my todo list. Else feel free to send me a pull request.

Regards
Vasilis

Reply (<http://blog.datumbox.com/using-artificial-intelligence-to-solve-the-2048-game-java-code/?replytocom=10015#respond>)




Vasilis Vryniotis (<http://blog.datumbox.com/author/bbriniotis/>)

Thanks Athi,

I made few really minor modifications of your fix and I posted it online. Thank you very much for spotting the issue and for contributing the solution.

Vasilis

Reply (<http://blog.datumbox.com/using-artificial-intelligence-to-solve-the-2048-game-java-code/?replytocom=10024#respond>)




Vasilis Vryniotis (<http://blog.datumbox.com/author/bbriniotis/>)

Hi Syed,

Yes that is correct the results of AI heavily depends on the depth that you explore. The search space is too large and thus we are forced to limit the depth of the search. This is configured by the depth parameter on the source.

Reply (<http://blog.datumbox.com/using-artificial-intelligence-to-solve-the-2048-game-java-code/?replytocom=10076#respond>)



Vasilis Vryniotis (<http://blog.datumbox.com/author/bbriniotis/>)

Hi Si,



July 2, 2014

I have not worked much again on this project. It meant not to take more than a weekend. There are various heuristics suggested (I have a link on the post to most of them). I proposed using a variance-clustering approach which seemed to work well. I'm sure that if you are interested in solving the problem faster, there are too many other things that you can try.

Best,
Vasilis

Reply (<http://blog.datumbox.com/using-artificial-intelligence-to-solve-the-2048-game-java-code/?replytocom=10084#respond>)



March 29, 2016

Chuck Fannin

Thanks for publishing this website – your development of software and sharing of knowledge is priceless for those of us still learning.

With that said, the algorithms used to solve 2048 are not really AI are they? These seem more like brute force algos that have been shortened because the problem space is too big (I believe minimax has been used in game theory for years – great for solving reasonable sized problems, but not AI.) I recognize there are many definitions of AI, but in this instance the algorithms are not learning from and/or tuning themselves by playing the game. My goal is to create a parameter space and have the machine tune the parameters by playing the game and learning its mistakes.

Thanks again for your website.

Reply (<http://blog.datumbox.com/using-artificial-intelligence-to-solve-the-2048-game-java-code/?replytocom=17400#respond>)



November 24, 2016

Ed

Here is your AI implemented on an EV3 (running leJOS) to win the game.
<https://www.youtube.com/watch?v=NNmHRbG5ulw> (<https://www.youtube.com/watch?v=NNmHRbG5ulw>)
(Depth is 5 for the first 200 moves, then 6 after that. It didn't win the first time but it was successful on the 2nd attempt.)

Reply (<http://blog.datumbox.com/using-artificial-intelligence-to-solve-the-2048-game-java-code/?replytocom=18019#respond>)

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Captcha *

Time limit is exhausted. Please reload the CAPTCHA.

six

-

four

=



Post Comment

Popular Posts Recent Posts



(<http://blog.datumbox.com/how-to->

build-your-own-twitter-sentiment-analysis-tool/)

How to build your own Twitter Sentiment Analysis Tool (<http://blog.datumbox.com/how-to-build-your-own-twitter-sentiment-analysis-tool/>)

September 2, 2013



(<http://blog.datumbox.com/10-tips-for-sentiment-analysis-projects/>)

sentiment-analysis-projects/)

10 Tips for Sentiment Analysis projects (<http://blog.datumbox.com/10-tips-for-sentiment-analysis-projects/>)

September 9, 2013



(<http://blog.datumbox.com/developing-a-naive-bayes-text-classifier-in-java/>)

Developing a Naive Bayes Text Classifier in JAVA (<http://blog.datumbox.com/developing-a-naive-bayes-text-classifier-in-java/>)

January 27, 2014



(<http://blog.datumbox.com/using-artificial-intelligence-to-solve-the-2048-game-java-code/>)

artificial-intelligence-to-solve-the-2048-game-java-code/)

Using Artificial Intelligence to solve the 2048 Game (JAVA code) (<http://blog.datumbox.com/using-artificial-intelligence-to-solve-the-2048-game-java-code/>)

April 7, 2014



(<http://blog.datumbox.com/new-open-source-machine-learning-framework-written-in-java/>)

source-machine-learning-framework-written-in-java/)
New open-source Machine Learning Framework written in Java (<http://blog.datumbox.com/new-open-source-machine-learning-framework-written-in-java/>)

October 19, 2014

Recent Posts

- Drilling into Spark's ALS Recommendation algorithm (<http://blog.datumbox.com/drilling-into-sparks-als-recommendation-algorithm/>)
- Getting the GPU usage of NVIDIA cards with the Linux dstat tool (<http://blog.datumbox.com/getting-the-gpu-usage-of-nvidia-cards-with-the-linux-dstat-tool/>)
- Datumbox Machine Learning Framework version 0.8.0 released (<http://blog.datumbox.com/datumbox-machine-learning-framework-version-0-8-0-released/>)
- Datumbox Machine Learning Framework 0.7.0 Released (<http://blog.datumbox.com/datumbox-machine-learning-framework-0-7-0-released/>)
- Datumbox Machine Learning Framework 0.6.1 Released (<http://blog.datumbox.com/datumbox-machine-learning-framework-0-6-1-released/>)

Archives

- February 2017 (<http://blog.datumbox.com/2017/02/>)
- January 2017 (<http://blog.datumbox.com/2017/01/>)
- March 2016 (<http://blog.datumbox.com/2016/03/>)
- January 2016 (<http://blog.datumbox.com/2016/01/>)
- May 2015 (<http://blog.datumbox.com/2015/05/>)
- November 2014 (<http://blog.datumbox.com/2014/11/>)
- October 2014 (<http://blog.datumbox.com/2014/10/>)
- July 2014 (<http://blog.datumbox.com/2014/07/>)
- June 2014 (<http://blog.datumbox.com/2014/06/>)
- May 2014 (<http://blog.datumbox.com/2014/05/>)
- April 2014 (<http://blog.datumbox.com/2014/04/>)
- March 2014 (<http://blog.datumbox.com/2014/03/>)
- February 2014 (<http://blog.datumbox.com/2014/02/>)
- January 2014 (<http://blog.datumbox.com/2014/01/>)
- December 2013 (<http://blog.datumbox.com/2013/12/>)
- November 2013 (<http://blog.datumbox.com/2013/11/>)
- October 2013 (<http://blog.datumbox.com/2013/10/>)