Simulation and Artificial Intelligence
Game of Life and Death Bot Report

Submitted for the MSc in
Advanced Computer Science

January 18

By
Alexander C Whitehead

Word Count: 2954

Table of Contents

1 Theory of AI Controller

There are three main ways in which the AI could be implemented to play the game of life and death.

Firstly, an AI could be written which attempts to find certain patterns in the layout of the cells on the board and either promote the formation of these patterns in their own cells or destroy these formations in the enemy cells. These formations include such shapes as; a block which is a stable pattern which will return back to a block upon killing one of its cells, the only way to combat this formation is to birth a cell next to it, a tub which is another stable patter however this pattern is relatively easy to kill by removing one cell from it, although this will cause it to spawn other patterns, a beehive which again is another stable patter which is easy to kill by removing one if its cells however this will cause it to create another four beehives and a blinker which is an oscillating pattern which will result in either an explosion or the formation of beehives when it is destroyed.
One way to implement this behaviour would be to sum a collection of squares on the board in a matrix, the result of which would equal a known value if it is one of these patterns or is close to being one of these patterns. Once the AI has identified these patterns all it would need to do would be to determine which pattern would result in the greatest positive outcome for itself and attempt to execute this pattern.

Secondly, an AI could be written which simulates each possible move that could be performed in a given turn or turns and executes the move with the greatest positive outcome for itself.
One way to implement this behaviour would be to step through every possible combination of killing and birthing cells storing the score of the highest performing move and then returning the highest scoring move when the time to calculate a move has expired.
In addition to calculating the current move the bot could also take into consideration what move the opposition is most likely to make in retaliation to your move and try to limit the damage that this move could inflict or to simulate multiple turns into the future as this could lead to the bot performing a combination play whereby it executes what may appear to be unideal moves now but which lead to a greater payoff in the future.

Thirdly, an AI could be written to use a neural network to learn how to play the game optimally. For this to work the AI would need to be trained on quality example games.
One way to implement this behaviour would be to implement a neural network which has the same number of input layer neurons as the board has cells, these input neurons would then be fed into the hidden layer of the neural network where calculations are performed on the specific combination of input neurons which results in an output of the most optimal move.

Finally, a bot could be implemented which uses a combination of the above techniques.

2 Analysis of AI Requirements for the Game

There are advantages and disadvantages to all of the implementations mentioned above.

Firstly, the main advantage of the pattern recognition AI is that it is incredibly simple to implement, none of the calculations required to recognise the patterns would be coupled to each other so even if only some of the patterns possible could be recognised it would be better than playing randomly. Also, these kinds of calculations can be performed incredibly rapidly by a computer and very few of them would need to be performed per turn thus the bot would return a move very quickly and there would be little chance of the bot timing out.
A disadvantage of this bot is that it has no real way of determining if what it is doing will result in it winning the game, all it is trying to do is make patterns it will not ever attack the opposition with the intention of killing all of their cells and as such many games with these types of AIs result in stalemates.

Secondly, the main advantage of the tree traversal AI is that if the simulations which determine the optimal move could be performed quickly enough it would be technically possible the calculate every possible move which could be made in the game and as such the AI could technically make the best possible move in every situation, this is referred to as a game being solved, games such as connect four and noughts and crosses have already been solved in this manner.
A disadvantage of this kind of AI is that the calculations used to determine the optimal move are incredibly resource intensive with up to 1.95398e+12 individual calculations having to be performed just to calculate the move for one turn, if the AI wanted to recursively check the next move too 1.95398e+12 * 1.95398e+12 calculations would have to be performed. Obviously with current hardware this number of calculations cannot be performed in a reasonable time.

Thirdly, the main advantage of a neural network is that it could attain the same or better performance as the tree traversal AI without having to perform close to the same number of calculations, this is because the hidden layer of the neural network has already been tweaked to the point where the output from this layer is the same as the tree traversal AI before it gets into a game. However, obviously this means that the neural network does not really have any understandable logic to it and neither the neural network nor the people who created it understand why it performs the moves that it does, this makes neural networks incredibly difficult to debug.
Interestingly neural networks are used extensively in research and industry fields to solve AI problems, for instance the Alpha GO AI which recently beat the world’s greatest go player was a kind of neural network using reinforcement learning and deep learning.
A disadvantage of this kind of AI is that they are incredibly difficult and time consuming to program mostly for the reasons mentioned above but also because the performance of the neural network is directly correlated to the amount of and quality of the training data it receives, it would be incredibly difficult to acquire the amount of training data required to train a neural network to play the game of life and death.

3 Design of Game of Life and Death AI Bot

The design chosen for this AI was a version of the tree traversal algorithm called the Monte Carlo tree search.

In order for the AI to calculate the next move it first simulates the board state if it were to pass, this board state is stored and then the bot simulates every outcome from killing every cell one by one on the board, these outcomes are compared and the highest scoring one is retained, next for each dead or empty cell on the board the AI simulates the outcome of killing every combination of two of its own cells to birth a new cell in that place, the highest scoring combination is checked against the previously highest scoring move and the resultant highest scoring move is retained.
After the highest scoring move is determined this move is then returned to the game being played.

In order to make the bot more competitive it should simulate multiple turns into the future including enemy moves, to do this before checking the score of each move against each other it should pass the resultant board state from each move to another AI which performs the same logic as the first AI but it tries to maximise the score for the opposition, the board state resulting from this can in turn be passed to yet another AI which again tries to maximise the score in favour of the initial AI, this can propagate indefinably or until a game winning combination of moves can be determined.

To increase the speed of the calculations for each move only the squares which differ from a passing move need to be resimulated for each new move. E.g. if the passing board state has already been determined this can be passed to the killing algorithm, this algorithm can take the passing board state and initial board state, simulate the eight cells around the cell it intends to make a move on using the initial board state and then combine this new board with the passing board state to get the killing board state.
This can also be implemented on the birthing algorithm although it would have to be executed three times rather than just once, once for every cell that is being changed by the current move.

An alternative way to increase the speed of the calculations is for the bot to ignore all dead cells which share multiple neighbouring dead cells, if these clusters of cells can be tracked accurately they can be ignored from future calculations as they can be deemed to not update in the current simulation step.

Also, rather than implementing a naïve tree search alpha and beta tree search pruning techniques can be applied to the algorithm. For instance, if a node on the tree scores vastly higher in the oppositions favour that branch can be deemed to be ignored, this is because the branches which score favourably in the oppositions favour yield the same value as another subtree or worse, and as such will probably not influence the final result in the AIs favour.

4 Implementation

In order to simulate each turn a method was implemented which takes in the start board state and for each possible move, iterates through each cell checking each of its neighbouring cells in turn and incrementing a variable which tracks the number of live neighbours if a neighbouring cell is alive, it then also checks if the neighbouring cell is a cell belonging to the opposition and increments another variable if it is.
Once this has been performed the state of the cell in the next simulation step is determined from its number of neighbours, if the cell is alive and had fewer than two live neighbours it is changed to being dead, if the cell is alive and has two or three neighbours its current state is copied, if the cell is alive and has more than three neighbours it is changed to being dead, if the cell is dead and has exactly three neighbours the second variable is then checked, if it is greater than one the cell will be born as an enemy cell, however if it is less than two then the cell is reborn as a cell of the current AIs type.

The score is tracked through the program by passing a temporary score to each simulation instance, the AI first simulates the turn as mentioned above and then adds together every instance of a cell belonging to itself and a cell belonging to the enemy, the number of enemy cells is then taken away from the number of friendly cells. If this score is greater than the previous score this score now becomes the current score and the output string is updated to contain the higher scoring move.
If multiple simulations are to occur into the future the score for each node is determined by the deepest simulation as this represents the move that the AI would like to work towards. An advantage of this scoring technique is that it is not required that the AI needs to know when to attack or defend, it is only trying to have more cells than its opposition, which naturally sometimes means defending and sometimes attacking, it is whichever would have the greatest outcome for the AI.

There were a lot of issues and deviations from the original design while implementing the bot. For instance, the time limit to make a move in the game is rather unforgiving and as such it was difficult to calculate even the first set of moves every turn without running out of time never mind simulating multiple turns. If every possible move was evaluated naively for one turn it was still taking upwards of three seconds per turn and considering that the max time limit to play one hundred turn was 10 seconds this play style was not sustainable.

In order to simulate multiple turns an enemy AI has also been implemented which can be instantiated in the initial AI. This AI has the same capabilities as the initial AI, however because of the time constraints of each move the AI only simulates passing a turn, this does improve the abilities of the initial AI still however. This is because there will always be two simulation steps between the AIs current move and its next move, in game, so by passing on the enemies turn it allows the initial AI to fully simulate the outcome of its move up to the next time it makes a move.

A timer function was implemented in the AI which tracks the time from when the AI receives the command from the game to make a move up to current time, if this timer ever exceeds the time per move received from the game plus one hundredth of the overall time to play the game the AI returns the current highest score, this is because sometimes games can end in stalemates where both players have blocks of four cells together, in this situation it is impossible to win but it is also impossible to lose via time out because of the timer. It is better to draw than to lose so the bot must always have enough time left in its bank to complete a full one hundred round game and the timer ensures this to be true.

5 Analysis of Performance

Overall there are advantages and disadvantages to the performance of the bot.

Firstly, because it is impossible to naively check every possible move for every turn the bot must make some compromises, for instance rather than checking every cell including its own cells when checking the killing algorithm the bot only checks to kill enemy cells, this is because although there may be some instances where killing your own cells is advantageous the vast majority of the time it is not and as such there is more worth in banking the extra time rather than checking every possibility.
Also the bot cannot check the combination of every birth cell to sacrifice cell as it takes too long so the optimal birth cell is first determined like the killing cell is and then this one birth cell is paired with every combination of sacrificial cells. This is also advantageous as it speeds the AI up so much that it allowed the AI enough time to check the outcome of the enemy passing the next move mentioned above.

Secondly, because the AI was not taught pattern recognition it can end up in the situations mentioned above whereby it has to pass every turn when in a four block stalemate with another AI, it would have been advantageous to implement basic pattern recognition so that these situations can be noticed well in advance and dealt with, this would ensure a much higher win rate as most losses observed during testing occurred because of this.
However, it could also be said that if pattern recognition was implemented it may slow the AI down enough so that it could no longer fully check every move so it is very much a catch twenty two situation.

Thirdly, an advantage of the AIs design is that is it highly modular, if pattern recognition was to be added to it in the future or the algorithm was to be optimised further so as to add deeper tree walks it would be very easy to implement these changes as the AI has been developed with this in mind and coupling between classes has been kept to a minimum.

If another AI was designed to play this game based on data from the first AI a better approach would be to design a machine learning AI in the same style as Alpha Zero, this AI would engage in reinforcement learning by playing games against itself until the AI is capable of anticipating its own next move and how these moves could affect the game’s outcome.
This is a better approach than the one mentioned above where the AI observes games to learn as the AI generates its own material to learn from and as such it is possible for the AI to learn endlessly.
Obviously one issue with this plan would be that a very advanced computer would be required to accelerate the AIs learning by playing multiple games in quick succession.

As of writing this report the bot produced for this project is ranked first in the Hull game of life and death leader board and is ranked twenty fifth in the global game of life and death leader board, the bot was ranked as high as seventeenth in the global game of life and death leader board.