Stratego AI – Nathan Krummen, Adam Miyashiro, Tian Jin

Introduction

Stratego is a board game, which presents itself as a strategy game similar to Chess. However, it has many components that add almost insurmountable complexity beyond what is required for a typical board game AI. Players have access to very little information about their opponent’s pieces, which makes the most basic premise for any Stratego AI the ability to soundly deduce and guess at the true value of an opponent’s pieces. Overall, a Stratego AI will need to implement multiple components of AI to replicate intelligent human play. In this paper we will detail 2 approaches we attempted in creating an intelligent AI. One being a statistical approach which worked on cost minimization and damage maximization, the other worked on using random attribution to reduce the problem into a state where a chess-like solution of Minimax with a static state evaluator as a heuristic could be implemented.

Description of game

Board

Common Stratego, is a full board game on a 10X10 matrix, with piece set of 6 Bombs, 8 (2)s, 6 (3)s, 6 (4)s, 5 (5)s, 5 (6)s, 4 (7)s, 2 (8)s, 1 (9), 1 (10), 1 Spy, and 1 Flag. These pieces are placed within a players’ respective set area, which is their half of the board, the first 4 rows on their side, with a restriction of the middle 2 rows. All pieces are hidden from the opponent, but visible to their player. In the version implemented in our solution, we chose a variant known as Stratego 10. Which has a smaller 8X8 matrix board to promote collisions between pieces, and only has the piece set of 2 Bombs, 2 (2)s, 2 (3)s, 1 (9), 1 (10), 1 Spy, and 1 Flag. This variant of the game adds a higher value to the minimization of piece loss, focuses moreso on positioning, and is an overall quicker game to play in order to add efficiency to our AI.

Piece Movement

Bombs and Flags are not allowed to be moved the entire game, once set at the onset of the game, they cannot be moved. (2)s are allowed to move in the same manner as Rooks or Castles in chess. Allowing them to move unlimited amount either vertically or horizontally each turn. All other pieces are allowed to move a single space each turn either vertically or horizontally. Diagonal movements are not allowed in this game.

Collisions

When a player’s piece comes into contact with the opponent’s piece, defined as moving them onto the same square, the value of the two pieces are revealed, and the piece with the higher value wins. There are exceptions to this rule, in that bombs will against all pieces with the exception of the (3), (10)s can be killed when a Spy attacks them, but cannot be killed when a (10) attacks a Spy. All pieces can kill the spy when they attack it. Ties between pieces result in both pieces being killed.

Winning

A win is scored, when a player comes into collision with the opponent’s flag. A tie is scored, when both sides have no moveable pieces remaining. A win is scored, if one side is the only side which has moveable pieces remaining. Note: As this is commonly disputed between Stratego players, when a player has killed all of an enemies (3)s, and their flag is protected by bombs, the opponent can still win by destroying all of the player’s moveable pieces.

GUI

In an effort to maximize time spent on the AI component, we strove for a bare bones GUI which ran in the command line. When the board is printed, each row is printed from 0 to 7 with each column labeled alphabetically from A to H. Each space on the board is printed as ‘\_\_\_\_\_\_’ if the space is empty, ‘HIDDEN’ if the space is occupied by a blue piece, or ‘Name(RANK)’ if the space is occupied by a red piece. On either side of the board, are the graveyards for each team. The graveyards list the pieces for each team that have been removed from the board. When a human is playing, a move is entered in as column, row (the space your piece is in) to column, row (the space you wish to move your piece to) e.g. ‘A4 B4’, ‘C7 C6’.

Approach #1

Approach #1 worked on the basis of using each pieces movement history within a given game, and comparing those numbers against presumed behavior for each level’s piece. This history, consisted of 5 attributes, defined as:

1. Total number of moves
2. Total number of retreats. Which is defined as the number of times an enemy piece is in a position to attack one of your pieces, and it fails to do so.
3. Total number of double moves. This worked as a True/False indicator for a (2) piece, since only (2)s can move more than 1 space at a time.
4. Consecutive moves. Meaning how many consecutive times did a single piece get moved by an enemy player.
5. Sideways moves. Number of times a piece was moved either left or right.

It was our original assumption that each piece had a certain behavior within the bounds of a dominate strategy. Spys are kept in the back, and move infrequently, and usually only after the (10) has been seen. (10)s move frequently, and have high levels of sideways movement as they position themselves to avoid Spy traps. (3)s have very little sideways movement, as they often streak across the board to disarm bombs in late game, etc. A player playing by these dominate strategies would be easy for us to deduce using this statistical prediction.

Each allied moveable piece would hold a matrix, which would only hold values across its diagonal. At the onset of each turn, the AI would calculate the %s associated with each enemy piece’s movement history, store this in vector form, and feed it into each allied piece’s matrix. Meaning, that for some given piece on spot A6, Each allied piece would be fed the following vector: {10%, 15%, 10%, …} which would correspond with {10% the piece is a 10, 15% the piece is a 9, 10% the piece is a 3, etc.}. These predictions were universal for each allied pieces’ matrix. On the horizontal vector for each allied pieces’ matrix, would be calculated functions of harm that piece had towards that enemy piece. In words, this harm value factored “Given that that piece is a 10, given its distance from me, what is the harm to me?” These harm values were crossed with the horizontal percentages, and a diagonal matrix was produced. To put this into example, let’s say that an enemy piece is in front of an allied (3). The allied (3) calculates that that piece has a 10% of being a (10), and the harm value calculated for it, given that it is indeed a 10, is 150. The values when crossed equals a harm value of 15. Now, let’s say that that same enemy piece has a 15% chance of being a (2), and a harm value, given that it is indeed a (2), is -200. Given these values, we yield a cross of -30. Each moveable piece would calculate these diagonal matrices, then run upon those matrices, looking for a move that either avoided a high-harm situation, or took part in a collision that yielded a negative harm value (an incentivization). This decision between high-harm minimization, and negative-harm capitalization was made by an “aggression decider”, which took into account advantage in the game (who had more pieces on the board) and a random element (to add unpredictability in play).

Overall, I believe our approach #1 would touch the ceiling in terms of acceptable run-time, and after multiple rounds of statistical tuning, a realization came to us as a team – No amount of likelihood can be placed within a game where bluffing is a strategy is so highly rewarded. A (2) could behave and masquerade across the board, acting as a dominate strategy (10) and scare all of our pieces into retreat, or for our AI to have our (10) or spy, run across the board to try to combat a (2). In the end, a player is just as likely to play in a dominate strategy, as they were to bluff. No amount of statistical fitting and abstraction could yield us any significant accuracy in our guesses, and as our run-time approached infeasibility, we decided to take up a different approach.

Approach #2

When faced with any significantly difficult problem, abstractions can build their way so high that they yield confusion. When that rickety tower of assumptions, haphazardly constructed, inevitably falls. We can point our next attempt to the concept of Occam’s Razor. Solutions with fewer premises are stronger than solutions with many premises. This was the concept that yielded our next approach. Instead of basing the guess of each piece on a statistical hunch, let us randomly assign each piece a perceived value. As the game progresses, each piece will produce some kind of behavior – not in the sense we used before, in basing our guesses on assumed behavior, but by basing our guesses on definitional behavior. To clarify, assumed behavior is the concept that we assume a (10) has a high amount of consecutive moves and horizontal moves. Definitional behavior is the fact that bombs can’t move, (2)s can move multiple spaces in a single move, etc. Their behavior isn’t an assumption, it’s a guarantee. So, at the onset of the game we gave a random assignment of each enemy piece. Let us say, that some piece on a random square is assigned “Bomb”, and the first move is for this perceived Bomb to move forward, we know that our assignments are wrong. So on the onset of our next turn, we update our information (including that this piece cannot be a non-moveable piece) and generate random assignments until we find an assignment that fits our information (this piece is labelled something which is moveable). In addition to these moves, let us say that by collision, we discover that some piece on the left side of the board is the enemy (10), but, our assignment had labelled some random piece on the right side of the board as the enemy (10), we update our information, and scramble our assignments until we find a new assignment which does not violate the premises established in our information. We named this kind of perception, The “Musical Chairs”, as when each assignment has some premise violated, the chairs are scrambled until a moreso fitting assignment is generated. Within the first 10 moves of any Stratego 10 game, 4 pieces are likely moved. Meaning that in a set of 7 moveable pieces, 3 immoveable pieces, if we know 4 have moved, we have a 50% likelihood of correct guessing non-moveable pieces within the first 10 moves.

Provided our perception model of Musical Chairs, we have a resulting game board to the AI which has a perceived 100% certainty in the values it guesses. Meaning, we have reduced the complexity from a Stratego AI, into a state which is solvable by a standard game board AI method. We designed a static board evaluator, which was based on the following attributes:

1. Endgame, which tests if the enemy’s flag is on the board \* 5
2. Number of one player’s pieces – Number of other player’s pieces \* 1
3. Vertical Movement, the sum of the distance each allied piece had to the 0 row (the row behind the enemy) \*5
4. Miner distance to un-moved pieces. The sum of the distance of both miners to a random unmoved piece \*5

Using this evaluator on every node, each board ran a minimax algorithm with a depth of 3, and implemented alpha-beta pruning for efficiency. The overall run-time for each move was about 4-5 seconds. When 40 games were run using our AI vs Random opponent our AI won 68% of its games (won 34 games and lost 16).

Conclusion

Using the “Musical Chairs” perception model, we developed an intuitive guessing strategy that allowed for problem-set reduction, and the ability for us as a team to apply decision algorithms previously learned through the class. Stratego is one of the most, if not the most challenging board game to model an AI after. In the national Stratego AI championship, where the world’s top Stratego AIs compete against each other to declare the most competent AI, even the best AI, are barely able to compete with a competent human player. We believe attempting to tackle such a difficult concept, and arriving at an AI whose playstyle is intelligent and reasonably competent, is an achievement.