Senior Design Project: Stratego

Jeff Petkun (petkun@seas.upenn.edu)
Ju Tan (ju@seas.upenn.edu)

Faculty Advisor
Mitch Marcus
# Table of Contents

<table>
<thead>
<tr>
<th>Section Name</th>
<th>Page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>I – Abstract…………………………………………………………………………3</td>
<td></td>
</tr>
<tr>
<td>II – Game Description………………………………………………………….......3</td>
<td></td>
</tr>
<tr>
<td>III – Related Work…………………………………………………………………3</td>
<td></td>
</tr>
<tr>
<td>A – Partially Obstructive Markov Decision Process (POMDP)………………4</td>
<td></td>
</tr>
<tr>
<td>B – Using Monte Carlo Simulation to Overcome POMDP Issues………………5</td>
<td></td>
</tr>
<tr>
<td>C – How our Implementation is Different………………………………………..5</td>
<td></td>
</tr>
<tr>
<td>D – Stratego AI World Championship Tournament……………………………6</td>
<td></td>
</tr>
<tr>
<td>IV – Technical Approach……………………………………………………………6</td>
<td></td>
</tr>
<tr>
<td>A – Architecture……………………………………………………………………6</td>
<td></td>
</tr>
<tr>
<td>i – Game Engine……………………………………………………………………..7</td>
<td></td>
</tr>
<tr>
<td>ii – Game Piece……………………………………………………………………..7</td>
<td></td>
</tr>
<tr>
<td>iii – OpenGL Window………………………………………………………………8</td>
<td></td>
</tr>
<tr>
<td>iv – CPU………………………………………………………………………………8</td>
<td></td>
</tr>
<tr>
<td>B – AI Bot……………………………………………………………………………8</td>
<td></td>
</tr>
<tr>
<td>i – Opening Board Position…………………………………………………………8</td>
<td></td>
</tr>
<tr>
<td>ii – Probability Distribution……………………………………………………..10</td>
<td></td>
</tr>
<tr>
<td>iii – Static Board Evaluator………………………………………………………11</td>
<td></td>
</tr>
<tr>
<td>a – What is a Static Evaluation Function………………………………………..11</td>
<td></td>
</tr>
<tr>
<td>b – Our Static Evaluation Function……………………………………………….12</td>
<td></td>
</tr>
<tr>
<td>c – Changing Weights for Static Evaluator Function………………………….12</td>
<td></td>
</tr>
<tr>
<td>iv – Handling Cycles……………………………………………………………….13</td>
<td></td>
</tr>
<tr>
<td>v – Minimax and Alpha-Beta Pruning…………………………………………….13</td>
<td></td>
</tr>
<tr>
<td>a – Minimax……………………………………………………………...…………13</td>
<td></td>
</tr>
<tr>
<td>b – Alpha-Beta Pruning…………………………………………………………….13</td>
<td></td>
</tr>
<tr>
<td>vi – Testing and Results……………………………………………………………..13</td>
<td></td>
</tr>
<tr>
<td>V – Conclusion………………………………………………………………………14</td>
<td></td>
</tr>
<tr>
<td>VI – Bibliography……………………………………………………………………15</td>
<td></td>
</tr>
</tbody>
</table>
I - Abstract

Games are a challenging marriage of artificial intelligence with human interaction. The goal of our project is to develop an open source desktop version of the popular board game Stratego with the ability to play against an AI bot. The task of developing an AI bot for Stratego is different from other popular games, such as chess, since in this game there are various uncertainties (e.g. each player does not know the placement of its opponent’s pieces until they are encountered during game play). This lends itself to the potential for bluffing. One of the challenges was to quantify this human characteristic of game play, bluffing, that is not present in other games such as chess, and develop an AI bot strategy that effectively accounts for unknown pieces while playing in real-time.

Our end goal for this project has been to create an open source AI bot to effectively play this game with a reasonable level of strategy against a human player. In our research we have tested a number of existing AI bots including Probe, the winner of the 2007 Stratego World Championship Tournament. We found most implementations, including Probe, to be beatable once we figured out their AI strategy. Also, current implementations were commercial and not extensible.

II – Game Description

Stratego is a game that consists of a 10x10 board with 80 game pieces. At the beginning of the game, players setup their 40 ranked pieces in the first four rows on their side of the board. Each player has 1 flag, 6 bombs, 1 Spy, and other differently numbered soldier pieces. The objective is to move pieces to capture the opponent’s flag. Every piece has a rank, and when two pieces collide, the piece with the lower rank is discarded with the exception that a Spy defeats a Marshal (#1). The soldier pieces can move one square per turn either horizontally or vertically except for the Scouts (#9), which can move multiple squares. When a soldier piece and a bomb collide the soldier is discarded unless the soldier is a Miner (#8), in which case the bomb is discarded. When the game starts, a player does not know how the other player’s pieces are placed on the board, but as they play the game, they learn through collisions.

III - Related Work

As mentioned in our abstract, most implementations of Stratego are commercial and closed source. Thus, one of our key initiatives was to provide the Stratego community with an extensible open source AI bot.

Developers of a commercial Stratego AI algorithm on Gamedev.net, a community game development forum, discuss a game board scoring policy that maps states and optimal actions. This policy requires the use of a stochastic state machine, in which the state machine is finite and transitions are probabilistic rather than deterministic. Other types of stochastic state machines are Dynamic Bayesian Networks and Markov Processes. On the development forum, the Stratego
game uses an algorithm that follows a Partially Obstructive Markov Decision Process, or POMDP (Timkin).

The POMDP algorithm implements a game board scoring policy that is used to compute three stochastic state maps: (1) AI’s influence map (player’s piece strengths), (2) opponent’s influence map, (3) probability distribution over board states for location of the flag. Next, the strategy used to capture the flag includes actions that maximize the influence of the AI in areas of the greatest probability of capturing the flag while diminishing the opponent’s influence in the player’s side of the board. The Alpha-beta pruning algorithm can then be applied to determine the best possible path.

A - Partially Obstructive Markov Decision Process (POMDP)

A POMDP is described as a specific case of Markov Decision Process. It maps the decision process for an agent that cannot directly observe the underlying states in the environment (Markov Decision Process). Thus, because the AI cannot observe the actual states of pieces, such as in Stratego, the decision mapping is based on an inference of the unknown states. These states are based on facts it knows about the environment and events that occur during the game play. This means the decision mapping (game scoring algorithm) would be a function of rules the AI knows about Stratego (e.g. the number of each pieces) combined with learned knowledge as the game play unravels such as when it learns the position of pieces it encounters during collisions.

The formal definition of a POMDP models the relationship between an agent and its environment.

- POMDP consists of a tuple of values (S,A,O,P,R)
- S is a finite set of states
- A is a finite set of actions
- P is a finite set of probabilities
- O is a finite set of observations
- R is the reward function where S,A \( \rightarrow \) R

For each decision, the environment is some state within the set S, and the agent will take some action A which will cause the environment to transition to a state S’ where the agent receives the reward R. The objective is to choose a series of decisions that lead to the state with the highest reward, ultimately capturing the opponent’s flag.
B - Using Monte Carlo Simulation to Overcome POMDP Issues

One of the problems with the POMDP approach is that it may be impossible to solve in polynomial time, thus making it unrealistic for real-time game play. While theoretically intriguing, it is not sufficient for our project purposes.

Thus, one possible solution is to replace creating each possible branch and then finding the most efficient path with simulating and sampling these different branches without having to generate a complete mapping of every scenario. An AI strategy implemented by computer scientist Rémi Coulom at the Université de Lille in France uses this technique. In particular, Coulom's program, Crazy Stone, won a gold medal for his GO Game AI at the 2006 Computer Olympiad in Torino, Italy (Borrell). Similar to Stratego, in the game of GO there are too many possible game scenarios to consider each in real-time. Also, in GO, it is not clear how to score each state of the board since it is difficult to tell who is ahead during game play. Thus, Rémi Coulom’s strategy uses the Monte Carlo Tree Search and Pattern Learning:

To evaluate a potential move, you simulate thousands of random games. And if black tends to win more often than white, then you know that move is favorable to black…Unlike traditional algorithms, the Monte Carlo approach is extremely easy to parallelize, so it can take advantage of the multi-core architecture of the new generation of processors…Because you can't sample every possible random game, the Monte Carlo algorithm can easily fail to find the best moves. For instance, if most of the random games resulting from a certain move are losses, but one is a guaranteed win, the basic algorithm would take the average of those games and still evaluate it as a bad position. Crazy Stone is clever enough to avoid this problem. When it notices that one sequence of moves looks better than the others, it tends to play it more often in the random games.

Remi Coulom, Wired Magazine
On his use of Monte Carlo Algorithm in his GO game AI

C - How our Implementation is Different

While the POMDP scoring model was intriguing, we wanted to keep our implementation running time as close as possible to real-time. With the POMDP model, it would be difficult to compute and store, ahead of time, the complete set S of finite states and set A of actions that could be encountered during game play since there are infinitely possible states and actions.

Our algorithm did away with tracking the AI’s influence map and opponent’s influence map. Instead, we tracked the probability distribution of pieces for each board state, and then utilized a static board evaluator along with Alpha-beta pruning to calculate the best moves at any given state. One benefit of this implementation was that it allowed us flexibility to adjust our static board evaluator throughout the different phases of the game: opening phase, middle phase, and end-game phase.
D - Stratego AI World Championship Tournament

The Stratego AI World Championship was started in 2007 and is open to any programmer whose application supports the Stratego API on the tournament website (StrategoUSA). It is hosted by Metaforge and the 2008 tournament took place in October. The site contains information about AI bots that competed in the competition and their related works. The winner of last year’s Computer Stratego World Championship tournament was Probe, a Stratego AI created by Imer Satz (Satz). It finished undefeated against the other contestants. Even though this AI Bot was undefeated, the forum of users who have tried to play against Probe said that they were able to beat it. Thus, there is significant room for improvement in the strategy of the AI bot, but most importantly, it shows how challenging this problem is since even the best bot is considered beatable by human players.

IV - Technical Approach

A – Architecture

![Architecture Diagram](image)

We have implemented a fully playable Stratego game with a GUI that allows a player to play against an AI bot or watch two AI-bots play against each other. In the Player vs. CPU mode, the human player is initially prompted to configure their board pieces. Next, the computer loads a board configuration from its playbook. During game play, the Game Engine is tasked with making sure moves are legal. If so, the Game Engine informs the UI to update the board and then prompts the AI bot for its move. The players alternate moves until one of the flags is captured.
Every time a move is made, the Game Engine sends the CPU bot the results, which allows the CPU bot to update its view of the game. We also implemented CPU vs. CPU mode, CPU vs. Random Bot mode, and CPU vs. Single Ply Bot mode for testing purposes. Using these three testing modes, we were able to evaluate the progress of our AI algorithm and tweak them.

One of the technical challenges we encountered was precisely defining the interface between the CPU and the GUI such that it will be easy to swap in new versions of the CPU to test out in our game. Each component of the game is separated such that future students can easily create new AI bots in order to improve upon our results.

Another technical challenge was to map bitmap images as our game pieces. We collaborated with an industry digital artist to create visually appealing game pieces, but had difficulty mapping them to our board. In an attempt to solve this, we reached out Dr. Badler, the Penn computer graphics professor, who pointed us to a package called Simple OpenGL Image Loader (SOIL) to load the images onto the board.

Figure 2: Before (Left) and after (Right) using SOIL to load images

i – Game Engine

The game engine initializes a new game when the user chooses a game mode by resetting the data structures representing the board and game pieces. Throughout the game, it keeps track of these data structures and updates them every time a move is made or a collision occurs. It sends information about the player’s moves to the glWindow so that it can update its picture of the board. In any mode involving the CPU, the game engine sends each move to the CPU so that it can update its view of the game.

ii – Game Pieces

Each instance of the Game Piece data structure contains information about a piece on the board. It contains the current location of the piece, the type of piece, and which player’s piece it
is. Both the Game Engine and the CPU classes have instances of Game Piece which they use to keep track of the current state of the board.

### iii – OpenGL Window

The OpenGL Window is used to display the current state of the board to the user. The user then interacts with the game by clicking on the OpenGL Window to make moves. The OpenGL Window sends information to the Game Engine about the moves that the user attempts to make as well as the initial board setup that the user picks.

### iv – CPU

This is where we implemented our AI bot which includes initial board setups, Alpha-beta pruning, static board evaluator calculations, and probability computations (described below). Similar to the OpenGL Window, every time the CPU makes a move, it sends an update to the Game Engine which verifies that the move is legal and updates its state of the game.

### B - AI Bot

Our CPU class contains our AI bot. It maintains its view of the board by keeping track of its pieces and its opponent’s pieces throughout game play. It also contains a probability distribution for each of the opponent’s pieces which it uses to figure out the risk and reward of attacking any given opponent’s piece.

### i - Opening Board Positions

When a new game is started, the Game Engine prompts the AI to load its opening board configuration. We implemented 10 different starting board arrangements, each with a different strategy and game play style. The CPU randomly selects a configuration from its playbook library and is capable of mirroring the configuration randomly, doubling the number of possible board configurations a user can play against.

Initial board positions are a crucial part of winning in Stratego. This is because, unlike in chess, there is no single configuration for board setups. A large portion of our opponent’s strategy revolves around trying to predict where the flag is placed. In creating opening board positions, we took into account the following factors:

- **Flag Placement**
  - Is the flag placed somewhere difficult for the opponent to guess?

- **Bomb Placement**
  - Are bombs strategically placed to either protect the flag, control what channels the opponent pieces can pass through, or as decoys.

- **Mobility of Pieces**
  - Given the initial board setup, is it possible to quickly discover the opponent’s pieces and are there enough higher valued pieces defending the weaker ones. Are
the Miners (#8), which are needed to destroy mines, diversified across the board such that the opponent cannot quickly pick them off?

- Control Over Channels
  - In chess, whoever controls the center of the board has an advantage since they can get to any part of the board from the center quickly. In Stratego, there is an advantage in controlling where the opponent’s pieces are allowed to cross the board.

Here are some examples (Collins, 2008):

![Figure 3](https://via.placeholder.com/150)

**Figure 3: Opening Board Position 1**

In Figure 3, the player uses the bombs as a tactical way to block off the middle and right channels that the opponent tries to use to cross to the player’s side of the board. This forces the opponent to direct his pieces to cross through the leftmost channel. Since in the beginning the board is full of pieces, it will be difficult for the opponent to destroy the bombs unless he has a Miner (#8) readily available in his front line of offense.

![Figure 4](https://via.placeholder.com/150)

**Figure 4: Opening Board Position 2**
In Figure 4, bombs are used as decoys. For example, the placement of bombs scattered along the lower portions of the board makes it more difficult for the opponent to guess where the flag is placed. Also, Sergeants (#7) are strategically placed behind the bombs protecting the flag since once a Miner (#8) attacks a bomb, then the Sergeant can quickly counter attack and destroy the Miner.

Figure 5: Opening Board Position 3

In Figure 5, the player uses two layers of bombs to protect the flag. Also, the front lines of the middle and right channels contain Scouts (#9) which can quickly move across the board to discover the identity of the opponent’s pieces (Scouts can move multiple spaces).

ii – Probability Distribution

Every time either player makes a move, the result of the move and any collision that occurred during the move is sent to the CPU so that it can update its view of the board. These events cause the CPU to update its probability distribution, which stores the probability of each opponent’s piece being of a certain type.

Example 1: The first event that causes the probability distribution to be updated is when the CPU observes that the opponent has moved an unknown piece multiple squares. In this case, the opponent’s piece must be a Scout (#9). The CPU updates its probability distribution so that the piece that moved has a 1.0 probability of being a Scout and a 0 probability of being anything else. The probability distribution is also updated to reflect that all other pieces have a lower probability of being a Scout.

Example 2: The second event that causes the probability distribution to be updated is when a collision occurs. In this event, the Game Engine tells the CPU what piece type was involved in the collision, and if its value was not known before, the CPU updates its probability distribution accordingly.
Example 3: The third event that causes the CPU to update its probability distribution is when a piece that has never moved before moves for the first time. In this case, the piece that moved cannot be one of the bombs or the flag. Also, all other pieces that have not yet moved have a better chance of being a bomb or the flag. In a way, this is the most important piece of information, especially for the end game strategy where pieces that have not yet moved are attacked in the hope of finding the flag.

iii - Static Board Evaluator

The goal of Stratego is to capture the opponent's flag. In order to reach this goal, the game play involves many decisions including whether to attack an opponent's piece, defend when under attack, and sacrifice pieces. When humans play board games, they tend to think a couple of moves in advance. Advanced players are generally capable of thinking ahead many more moves than beginners. However, no human and few computer programs are yet capable of thinking ahead to the end of a game because there are too many possible combinations of moves. For example, in an interview with World Chess Champion Garry Kasparov by Harvard Business Review, Kasparov shares that "...after just three opening moves by a chess player, more than 9 million positions are possible" (Coutu, 2005). Thus, with these mind-blowing numbers of possible moves, game engines often use a static evaluation function as a short-term utility function that approximates the value of the current static board configuration.

a) What is a Static Evaluation Function?

A static evaluation function is used to estimate the value of a static board position. The main defining characteristic of it is speed, not accuracy. It is also often called the heuristic evaluation function.

Specifically, heuristic sums the various factors of any given board configuration. For example, Francois Dominic Laramee shares that in his chess programming evaluation function he considers the following factors (Laramee, 2000):

- Material Balance
- Mobility of Pieces
- Board Control
- Development of Pieces
- King Safety

After identifying the factors that the function should measure, assigning the weights for each feature is crucial. An example static evaluator function used by Laramee for this chess program is:

\[
\text{Position Value} = W1 \times \text{[Material Balance(board)]} + W2 \times \text{[Mobility(board)]} + W3 \times \text{[Board Control(board)]} + W4 \times \text{[Dev of Pieces(board)]} + W5 \times \text{[King Safety(board)]}
\]
b) **Our Static Evaluation Function**

**Stratego Board Value** =

\[ W1 \cdot [\text{Discovery}] + W2 \cdot [\text{Material}] + W3 \cdot [\text{Flag}] + W4 \cdot [\text{Distance between miners and stationary pieces}] + W5 \cdot [\text{Vertical Movement}] + W6 \cdot [\text{Distance to Unknowns}] \]

In our Static Evaluation Function, we use six components which include:

- **Piece Discovery**
  This emphasizes discovering the value of the opponent’s pieces. Moves that involve collisions are given a higher value in this component.

- **Material**
  This is the value of the CPU’s remaining pieces. Each piece is weighted by importance so that the CPU can conserve its more valuable pieces and discover the board with its less valuable pieces.

- **Safety of Flag**
  This is computed by looking at the value of the pieces that are within two squares from the flag. Having higher ranking pieces and bombs near the flag indicates that the flag is better protected.

- **Distance from Miners to Potential Bombs or Flags**
  Boards where Miners (#8) are closer to pieces that have not yet moved are given higher scores because Miners are the only pieces that can destroy bombs, which usually protect the flag.

- **Vertical Movement of Pieces up the Board**
  One problem we had with our AI was that the CPU would get stuck moving back and forth between two squares because the board evaluator determined that to be the best moves available. In CPU vs. CPU mode, sometimes both computers would do this which would lead to a stalemate. One part of our solution to this was to give slightly better scores to boards in which pieces move toward the opponent’s side of the board.

- **Movement Toward Unknown Pieces**
  This was also implemented to break up cycles. Boards in which pieces are closer to unknown opponent’s pieces are given a higher score.

Determining the weights of each factor required trial-and-error. The appropriate static evaluator balances simplicity with comprehensiveness. It leaves the process of choosing the best move to the Alpha-beta pruning algorithm.

c) **Changing Weights for Static Evaluator Function**

As the game progresses, we adjust the different weights of the different components. Early in the game we only use discovery, material, and flag safety. Later in the game we take distance between miners and stationary pieces into account. As the game draws nearer to its conclusion, we adjust the weights such that distance between miners and stationary pieces has a much larger impact and also take into account vertical movement and distance to unknowns. It was important not to weigh distance between Miners (#8) and stationary pieces too high early in
the game because this would lead to all of the Miners being defeated and no possible way to win.

iv – Handling Cycles

One problem we came across during development was that the AI would sometimes get stuck moving its pieces in a cycle. We used our static board evaluator (discussed above) as one method for addressing this problem. Another way we attempted to handle this problem was by having the AI maintain, for each piece, the amount of times that piece had visited each board position. If the piece visits the same space on the board too many times, we do not consider that position for its next move. This does not hurt the AI’s chance to find the flag since the flag piece is stationary and thus cannot be in a position that has already been explored. This also reduces the number of branches in the Alpha-beta pruning search tree, allowing the algorithm to search to a greater depth. In cases where the AI has no choice but to move to a space it has already explored many times, we allow it to do so.

v – Minimax and Alpha-Beta Pruning

a) Minimax

Minimax is a commonly used strategy for AI bots. When given an allocation of pieces on the board, it is used to choose the best move possible. It chooses moves that minimize the loss resulting from the opponent’s best possible move.

b) Alpha-beta Pruning

Since the combination of moves grows exponentially, Alpha-beta Pruning is used to quickly get rid of branches that lead to useless positions. Alpha-beta pruning reduces the number of nodes in the minimax search tree. Specifically, “it stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move” (Alpha-beta pruning). According to Garry Kasparov, a World Chess Champion, in an interview after his famous tournament with IBM’s Deep Blue, “…despite how fast Deep Blue can analyze chess positions, there are certain advantages that favor human beings. When a player starts to think about a move, an evaluative process kicks in, allowing the player to quickly eliminate a whole series of worthless alternatives-moves that he doesn't even have to consider because he knows they're irrelevant. The computer, on the other hand, has to process all choices sequentially without eliminating any possibilities in advance” (Hopkins, 1995). In our Stratego game, Alpha-beta Pruning acts as this “evaluative process” that Kasparov speaks of.

vi – Testing and Results

In addition to Player vs. CPU and CPU vs. CPU modes, we added two additional modes called CPU vs. Random Bot and CPU vs. One Ply Bot. The Random Bot moves its pieces randomly while the One Ply Bot uses the static board evaluator but only looks one move into the future. The best way to test the AI’s progress was to have it compete against these less sophisticated bots. In fifty simulations against the Random Bot, our AI won 36 times, never lost,
and tied 14 times. Against One Ply Bot, our AI won 28 times, lost 5 times, and tied 17 times. This shows that our AI’s strategy is far superior to both of these bots. One cause of ties was that our bot struggled to find the opponent’s flag when the opponent used an initial board setup that blocked the middle channels with bombs in the front row.

V – Conclusion

Implementing the game of Stratego with a playable AI turned out to be a very challenging and rewarding experience. Because we developed both the architecture for the game and the AI, we learned a lot about real game development. On the architecture side, the challenge was to design a configuration of components that interacted well together. The toughest technical problem was loading the images onto the board. On the AI side, once we developed the basic structure which did Alpha-Beta Pruning with a static board evaluator, we continually tweaked our AI to handle cycles, have a stronger end-game strategy, and better protect the flag. Overall, after testing against other bots, we conclude that our AI is a successful one capable of playing the game in a reasonably intelligent manner.
VI – Bibliography

Borrell, Brendan. AI Invades GO Territory. 19 9 2006. 22 10 2008


<http://www.azer.com/aiweb/categories/magazine/33_folder/33_articles/33_kasparovibm.html>.


StrategoUSA. 2008 Computer Stratego World Championship. 6 9 2008. 22 10 2008

