Learning Stratego

Michelle Chesley and **Coline Devin** and **May Lynn Forssen**
Harvey Mudd College
301 Platt Blvd
Claremont, California 91711

Problem

We plan to write a Q-learning agent that can learn to play the game Stratego. Stratego is a board game with two players. Each player has several pieces representing soldiers, bombs, and a flag. The objective of the game is to find the opponent's flag. The win condition is to either capture the opponent's flag or capture so many of their pieces that they can no longer make a move.

There are two phases to the game: the setup phase and that play phase. In the setup phase, each player sets their pieces up on their side of the board, in locations that they think will be most beneficial to the game. Then, in the play phase, each player may move only one of their pieces to a legal square. Our goal will be to implement an agent that can learn how to react to both of these phases.

The pieces are placed on the board facing away from one another so that a player cannot see what their opponent's pieces are, so it is a game where we have incomplete information about our adversary, which will make it an interesting artificial intelligence problem for us to solve. The different pieces have a different amount of strength associated with them, and this strength is only revealed to the other player when one pices attacks another. The goal of our project will be to make an AI that can successfully play Stratego on a human level.

Literature Review

To research methods for our project, we read several papers that described how to deal with having imperfect information about the world and playing against other agents.

Hasinoff

The paper "Reinforcement Learning for Problems with Hidden State" by Hasinoff describes ways of dealing with POMDPs (partially observable Markov decision processes): MDPs where some parts of the state are unknown. For reinforcement learning, Hasinoff explains that greedy Q-learning is non-optimal because of the unknown information: the values might never converge. The problem with memoryless learning (such as q-learning) that multiple states are grouped in the same observation (the paper gives the example going through a maze but only knowing number of walls around you. Without memory, you might see taking a step as putting you in the same state as if you had not taken a step). In POMDPs, this problem is inherent because even with a perfect feature vector, some states are indistinguishable because of the hidden information. One way of distinguishing states is to give the agent memory.

Hasinoff describes Nearest Sequence Memory (NSM). NSM is a method to determine which states among those with the same feature vector are actually the same state. It is based off assumption that states that were reached by a similar history of experiences are more likely to be the same state. The algorithm is as follows:

1. Record the history of experiences (action, observation, and reward or last n experiences).
2. Using the distance metric, find the k previous states that are closest to the current state.
3. Approximate Q-values for current state by averaging values for the k states.
4. Pick an action, then add the experience to the history.
5. Apply the standard Q-learning update to all states involved.

This paper is relevant to our project, as we are creating a learning agent to learn a game with imperfect information.

Hu and Wellman

In the paper "Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm", Hu and Wellman describe a way for reinforcement learning to treat other agents differently from the environment. Usually in reinforcement learning, an agent treats other agents the same way that it would treat elements of the environment. They assume that agents have "incomplete but perfect information" about each other, which means that they do not know the reward function for the other agent but can see their immediate rewards and actions. An agent cannot just maximize its own

Q-values, since those values depend on the actions of the other agent. They propose that each agent should maintain two tables of Q-values – one of its own, and one of its opponents'. The Q-values for the agent are updated by:

$$Q^1_{t+1}(s, a^1, a^2) = \\ (1 - \alpha_t)Q^1_t(s, a^1, a^2) + \alpha_t \left[r^1_t + \beta\pi^1(s')Q^1_t(s')\pi^2(s')\right]$$

The Q-values for the opponent are updated by:

$$Q^2_{t+1}(s, a^1, a^2) = \\ (1 - \alpha_t)Q^2_t(s, a^1, a^2) + \alpha_t \left[r^2_t + \beta\pi^1(s')Q^2_t(s')\pi^2(s')\right]$$

When the game is a zero-sum game, we only need one Q-table, since a gain for one agent means a loss for the other. Therefore, the Q-learning algorithm will be the following for a zero-sum game:

$$Q_{t+1}(s, a^1, a^2) = \\ (1 - \alpha_t)Q_t(s, a^1, a^2) + \\ \alpha_t \left[r_t + \beta \max_{\pi^1(s')\in\sigma(A^1)} \min_{\pi^2(s')\in\sigma(A^2)} \pi^1(s')Q_t(s')\pi^2(s')\right]$$

This is different from normal Q-learning, in that we are accounting for the policy of the second agent.

This paper is relevant to our project, as we will be playing two agents against each other, and they will therefore need to keep track of one another. Stratego is a zero-sum game, so we would only need to keep track of one table of Q-values, rather than two.

Littman

Many reinforcement learning algorithms assume that the agent's environment is stationary. "Markov games as a framework for multi-agent reinforcement learning", by Michael L. Littman, looks at applying reinforcement learning to two-player zero-sum games using the Markov game framework instead of MDP. Instead of just having one set of actions for each state, a Markov game has a collection of action sets, one of each agent. In an MDP, there is always an undominated policy for each state.

In a Markov game, however, there may not be, because the result of any action depends on the opponent's action. Game theory solves this by using minimax. Find an optimal policy using value iteration is essentially the same for Markov games as it is for MDPs. You just consider two moves at a time (yours and your opponent's) instead of just one. Similarly, Q-learning is easily adaptable from Markov games to MDPs. Again, you just consider both your move and your opponent's, instead of just yours. This algorithm is called minimax-Q, since it is Q-learning using minimax instead of just max.

This paper is relevant to our project, as the environment in our game will not be stationary, since the other agent will be moving pieces as well, and the results of our actions could depend on the opponents actions. Therefore, we could use this mathod to maximize score against our opponent by using minimax Q-learning to take the actions of the opponent into account.

Method

We plan to use reinforcement learning to create our AI. We will use feature-based Q-learning. Because the state space of Stratego is so large, we will not be able to actually explore or store it all. To deal with this, we will need to determine relevant features to represent search states. We have decided that some of those relevant features include:

- The number of pieces the agent has left and the reciprocal of the number of pieces the opponent has left
- The sum of the ranks of the agent's remaining pieces and the reciprocal of the sum of the ranks of the opponent's remaining pieces
- The number of bomb diffusers that the agent has on the board
- The number of bombs that the agent has on the board
- The distance between the agent's flag and the nearest enemy piece
- The sum of the rows of all of the agent's and opponent's pieces on the board
- The number of squares adjacent to the agent's flag that are occupied by enemy pieces.

These features should accurately capture the important aspects of a given state, and give our agent a way to accurately evaluate the game states that it is in.

We have created a model of the game in Python. The model will be able to play two AIs against each other, or play an AI against a human opponent. We will train our agent by running two versions of the agent against each other, each using the features mentioned above to update their Q-values and improve their playing strategies.

When we have finished the project, we will have a Q-learning agent and a human agent that takes user input so that a human can play the game against the computer. To make the agent possible, we will also have a feature extractor and game state representation.

Current Work

At this time, we have completed a working Stratego game, as well as several non-learning agents. The game is text-based, using a grid system based off of the provided grid-world code from Project 3, and tracks the position and rank of every piece on the board. To preserve the unknowns in the game, when agent asks the system for the state, the system only gives it the location of the opponents pieces and not the identities. An agent can ask the game state for a list of all the legal actions that it can perform, and when it

chooses one, the game state can update itself to take the new action into account. The game state can also check whether the state is terminal.

We have decided to play the game using an 8 by 8 grid as the board with 10 pieces per player rather than a 10 by 10 board with 40 pieces per player as in the classical game, because this smaller board and piece selection is an option available in the online game, and it will have a smaller state space and number of possible actions than the larger board. We will still be able to play our game and test it against online players, but the time our agent will take to learn to play will be reduced due to the smaller board size and fewer number of pieces.

The pieces are represented by their own class. They each have a rank, position, and agent index (which indicates which player the piece belongs to). Pieces with ranks of lower numbers are more powerful than those with higher numbers, and when two pieces fight one another, we have written an attack method to decide which piece is the winner.

We have written a feature extractor written that examines the game state and extracts its relevant features, which are described above. These features are returned in a list, and will be used by our learning agent to evaluate a given state.

We have also written a couple different agents. We have a random agent, that randomly picks a legal action to play. We have a human agent, that queries the user for a piece to move and a location to lose it to. The agent then checks that the move is legal. The human agent that we have allows a player to perform any legal move on one of their pieces during their turn. We have started writing a reinforcement learning agent, but have not tested it yet.

We have successfully run several games playing the random agents against one another and they finished successfully, usually taking several hundred turns to finish and with each player winning about half the time. We can also successfully play games using the human agent by manually giving input as to where each piece should move.

A difficulty we face at the moment is setting up the game. When playing Stratego, a large portion of the game is choosing where to place each piece before starting the game. So far, we have an algorithm for generating random setups, but we still need to implement reinforcement learning on the setup.

Metrics of Success

We will measure the success of our AI agent by counting how many times it wins or loses against a variety of opponents. We will first have the agent play against a random player. Once it can do better than random, we will use it to play against humans online, and against ourselves. To play online, we will make a version of the game where we can input the opponent's moves and our AI can tell us what move to make inresponse.

Regardless of whether the AI we write can meet these goals, we will learn how well reinforcement learning can work on a game with a state space as large as that of Stratego, as well as gain a lot of experience in writing artificially intelligent agents and figuring out which features of the world state would contribute most towards an artificially intelligent agent learning intelligent behavior.

Feasibility

We believe our project is feasible because, although the game of Stratego is complex and has hidden information, we are approaching the project in a fairly straightforward way. To get enough training time, we will train the agent against itself, a technique that worked very well for TD-Gammon**??**. As none of us are particularly skilled Stratego players, it should be possible for the agent to learn to play better than us and better than the random agent. Our stretch goal of having the agent play against (and beat) good players online will be harder, and its feasibility will be easier to determine once we have started training the agent. The AI agent on stratego.com reasonably good at the game, and was able to win against us.

Future Work

The main things we need to work on next are:

1. Write the feature extractor.
2. Test the Q-learning agent.
3. Figure out how to to make Q-learning.
4. Set up the system to have 2 agents learning at once.

Getting these things done will allow us to mostly finish the project. Additional tasks to do could involve creating a non-learning look ahead agent to train against or putting the agent online to have students play against it.

References

Hasinoff, S. W. 2003. "Reinforcement Learning for problems with Hidden State". Department of Computer Science, University of Toronto.

Hu, J and Wellman, M. P. "Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm". Artificial Intelligence Laboratory, University of Michigan.

Littman, M. L. " Markov games as a framework for multi-agent reinforcement learning". Department of Computer Science, Brown University.

Tesauro, G. 1995. "Temporal Difference Learning and TD-Gammon". Communications of the ACM. Vol. 38, No. 3.