Victorybot: An attempt at a full-featured Diplomacy (the game) AI

Benjamin Podgursky, Drew Scoggins

Vanderbilt University, Electrical Engineering and Computer Science Department

Abstract

In this paper we discuss the design and implementation of an AI to play Diplomacy, a strategically and tactically complex board game. The effectiveness of several search and state space pruning strategies are explored and their performance is compared. We also investigate the body of existing literature on multi-player search and other implemented Diplomacy AIs. We also discuss more complex aspects of Diplomacy, such as, diplomatic communications and probabilistically analyzing the behavior of other players. In our results section we compare the performance of four configurations of our bot.

Introduction

The game of Diplomacy is widely acknowledged as one of the most tactically and strategically complex board games, even for games with only human players.

A number of unique characteristics among board games makes Diplomacy a rich testing ground for the use of different forms of AI (Haard, 2004) as well. The game has a huge state space, even at the beginning or end of gameplay. There is no randomness to gameplay—all actions are deterministically defined by players' orders. All moves are simultaneous, with fairly complex rules for resolving the orders for all players. Most importantly, finesse in communication and negotiation is vital for successful gameplay—it is impossible to win without making (and breaking) alliances.

Luckily for our project, there is already a fairly active community developing Diplomacy AIs. The DAIDE gameplay and communication framework was designed as a common interface for different implementations to compete against one another. Section 2 describes this framework along with a basic description of gameplay. Section 3 will include discussion of other Diplomacy bots as well as other multiplayer game search strategies. The specifics of our implementation and other design decisions will be discussed in section 4. Section 5 and 6 will describe our experimental methods and results of our experiments. Section 7 will contain general discussion of our experimental results as well as the development cycle and various problems encountered during the project. Finally, section 8 will contain our conclusions of the project and final thoughts on the AI components involved as well as future work.

Description of the Task(s)

The primary goal of our project was to investigate how well traditional AI techniques would translate to the game of Diplomacy—of course, in the process we did also want to produce an AI comparable to other Diplomacy bots published in literature and played online.

The development goals of this project fell roughly into three tasks: first, develop an efficient and verbose state representation; second, develop AI search and space pruning techniques; and last, interface with the DAIDE test environment. In this section we discuss at a high level each of these tasks.

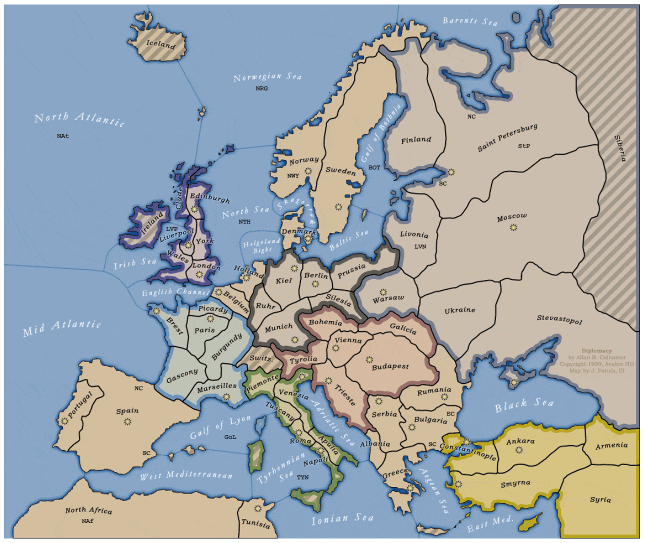
Overall Gameplay

This section gives a very abbreviated description of gameplay, because the full set of rules is longer than the page limit on this paper. For a more comprehensive description, please see (Avalon Hill).

The game of Diplomacy is a seven player, fully deterministic board game in which players attempt to conquer the Europe starting of 1900. Players take the roles of either France, England, Germany, Italy, Russia, Turkey, or Austro-Hungary. The map shown below is the classical Diplomacy map.

A territory on the map can be either a *land* or *sea* territory. Territories with small circles on them represent *supply centers*, the control of which counts towards victory. The only two unit types in the game are the *fleet* and the *army*. A fleet is able to reside on any sea territory, or a land territory that borders an ocean. In a turn, a unit is able to attempt to *move* to any neighboring territory it is allowed to occupy (ie, an army cannot move into the North

Sea.) Fleets also have the ability to *convoy* armies—one or more may form a link across an ocean that an army can



cross.

In addition to movements, holds and the convoy, a unit can also *support* another unit, by either supporting it holding or supporting its move into a neighboring territory.

The adjudication algorithm to resolve conflicts when two players try to move to the same location is fully deterministic—there is no dice rolling or use of cards to find the outcome of a set of moves. The figure to the right shows move resolution for some basic scenarios. In the top situation, the green player in B is attempting to move into the territory C. The purple player in territory is in turn C attempting to move into territory B. Finally, the pink player in territory A decides to support the movement of the green player into C. As a result, the green player is able to move into territory C, and the purple player there is forced to retreat into territory D. If instead the pink player had supported the purple player into territory B then the purple player would have moved into B and the green player would be forced to retreat.

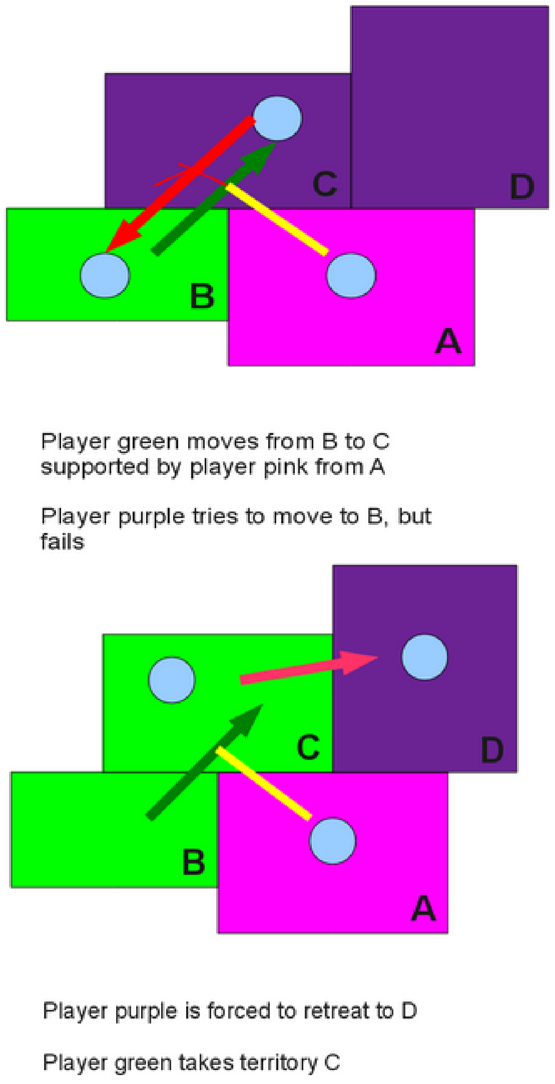
A player wins the game by gaining control of 18 supply centers.

AI Challenges

While this general concept of "moving units" is not unique to Diplomacy, the gameplay differs considerably from games traditionally studied by formal AI techniques:

First, Diplomacy is nearly unique in that movements are simultaneous. At each timestep, every player submits an order for each of their units; all orders are secret until all orders by all players are collected and revealed. When orders conflict with each other, the adjudication algorithm computes the resulting board state. Almost all studied games, from Chess to Go, involve sequential actions of single units.

Second, the order secrecy and movement support mechanics of the game make interaction and negotiation with other players a vital part of gameplay--it is almost impossible to win a game without working with other players. Few traditional AI players attempt to negotiate, because they don't have to—in Chess, there is no point in negotiating with your opponent.


C
D
B
A
Player green moves from B to C
supported by player pink from A
Player purple tries to move to B, but fails
C
D
B
A
Player purple is forced to retreat to D
Player green takes territory C

Last, because every unit on the board moves at every timestep, the game has an enormous search space and a very high branching factor—each board state has millions of potential child states. Later in the paper we talk about the algorithm strategies for pruning this space.

The combination of negotiation and uncertainty begs similarities between Diplomacy gameplay and planning under uncertainty; if a player can get a reasonable estimate of opponent move probabilities, it gives the player a big advantage. While we do touch on this in our system architecture, in general it was outside the scope of this paper and was left for later work.

Games with simultaneous moves is a model commonly studied in game theory, but generally on toy problems, with a focus on finding an equilibrium solution. We did not have time to look deeply into how game theory results could be integrated into our AI.

Interface

We decided to use the DAIDE communication framework to allow competition between different bots. A DAIDE server hosts games over a TCP/IP connection; bots compete by connecting to the server. The server is responsible handling all communications and adjudicating the results of the orders at each turn.

To help standardize communication between Bots (so they do not have to handle full NLP), the DAIDE server has a syntax which defines different levels of *press*. For example, at level 10 press, bots communicate to each other the desire to ally; at level 20 press, bots suggest moves to each other; press levels go to 130. By telling the server which press levels it is able to understand, it is easier to compare bots "on a level playing field."

Related Work

We look at two types of related work for this project; first we look at existing Diplomacy AI implementations and strategies. Second, we look at theoretical results in multi-player game search.

Diplomacy

The Diplomacy AI community has historically been fairly active, and there is a decently large body of existing AI implementations to compare against. (Webb & et, 2008) Almost all of these AIs use the DAIDE framework discussed above.

The first we looked at is dubbed DumbBot, designed by David Norman (Newbury). DumbBot, despite its name, uses a simple heuristic to generate high-quality moves. DumbBot simply goes over the map and assigns a to each territory based on a heuristic estimation of quality. These numbers have to do with who owns the supply center, or if it is neutral, as well as the strength of the power that holds the supply center. After the value of each node had been calculated, each unit at picks a territory that has the highest node value that it is adjacent to and attempts to move there. Although the heuristic is simple, it is quite effective at tackling the problem of estimating board and move quality.

Another bot that begins to incorporate communication is the BlabBot (Newbury). This bot builds on the DumbBot architecture, but adds crude level 10 press. The bot at the beginning of the game sends out a peace offer to all other bots. If all accept then it sends out a draw request to end the game. However, if some do not respond or respond in the negative then it will declare some nations as friend and others as foe. All friend nations have any the score of moves that move into their territory reduced significantly in order to remain at peace with them. One major flaw with BlabBot is that no attempt is ever made to ascertain the truthfulness of a proposed agreement. In other words, if a bot declared peace with BlabBot and then did not honor it, BlabBot would continue to honor the peace agreement (Newbury).

The arguably most advanced bot so far is Albert. Albert is a departure from the simple heuristic of DumbBot. The implementation is closed source, but documentation suggests that Albert attempts an actual game-search, rather than relying on simple heuristics.

Albert (Hal) spends time analyzing the best moves for other players and as it gathers more information attempts to change the probability of each move set. In this way it is able to iteratively improve its understanding of what moves others will do, and as a result select better moves for itself. Albert also contains the ability to run up to Level 20 press. It is both able to tell when agreements have been broken and has a built in deceitfulness factor that allows it to attempt to trick less sophisticated bots. Below is a brief summary of the three bots and the heuristic they use, the press level they communicate with and whether or not they use some sort of search.

	Heuristic Type	**Press Level**	**Search**
DumbBot	Simple scoring of nodes. Choose best nodes.	Level 0	No search
BlabBot	Simple scoring of nodes. Choose best nodes.	Level 10, but crudely	No search
Albert	Score nodes then generate opponent move sets. Then iteratively discover probabilities of move sets and change our moves accordingly	Plays a sophisticated Level 20 press.	Does minimal search in order to rate probabiliti es of enemy move sets.

Multi-player Games

Perhaps surprisingly, multi-player game search has not been extensively studied in an academic setting, especially compared to the exhaustively studied two-person game search.

The first, and standard, extension of the standard mini-max search technique to many players was the max-n algorithm [an algorithmic solution of n person games]. The max-n technique as described there looks at instances of multi-player search from more of a game-theoretic perspective. The max-n algorithm explores the entire game tree and locates an *equilibrium point*, where no single player can change their move to increase their move utility.

Other papers (Sturtevant, 2006) have extended the max-n algorithm by attempting to develop more sophisticated pruning techniques. The max-n search can be somewhat pruned on games with monotonic heuristics; we point to (Sturtevant, 2006) for examples. The game we are studying here, however, does not have any such heuristics.

Unfortunately, admissible deep pruning techniques in multi-player game search are more or less impossible to design; to prune a branch from a search node in multi-player search, we have to have certainty that the player making the choice at that node will not want to choose that branch, if they reach that node. However, we have no way of learning the utility of that branch to that player, without first expanding the branch! For a more formal description of this idea, we point to (Sturtevant, Last-branch and speculative pruning algorithms for max n , 2003)

System Description

Before describing the implemented system, we note that all of the source code for this project and the experiments is open source and available via http://code.google.com/p/victorybot/. The system modules described here match closely with the source code, although some details are of course omitted for brevity.

Design Goals

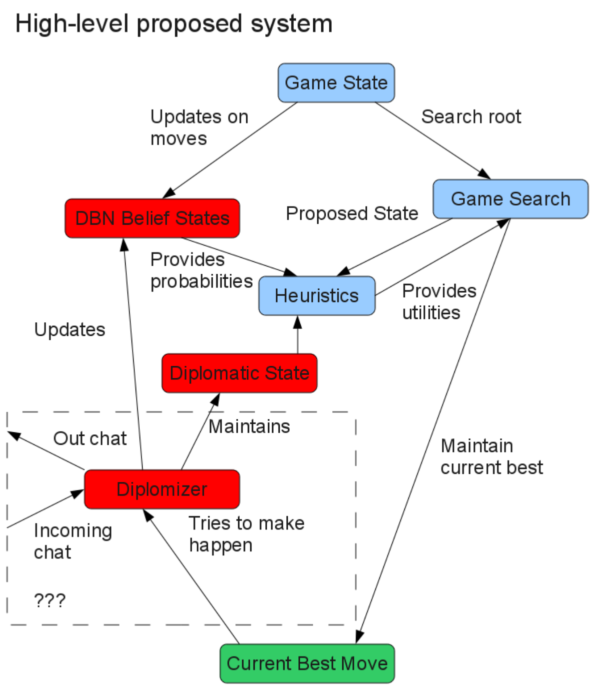
When designing the overall architecture for our system, we considered the following goals:

- Clarity of design—since the major goal of this project was to study new or existing AI techniques, we wanted to design a system with some degree of independence between the algorithms and the domain.
- Modularity—we wanted the major components of our architecture to be interchangeable, so we could easily compare different strategies.
- Ease of implementation—given the relatively short development timeframe, we wanted to be able to focus on AI strategies, not debugging.
- Work with existing approaches—instead of solely setting out to dominate humans at the game, we wanted a system which could be used in conjunction with human expertise, potentially as an expert system.

Original Architecture

While software design and architecture is not particularly exciting, the following section helps make clear where the algorithmic techniques we developed fit into the overall system.

Our proposed architecture is shown below. Components in blue are the components which we were able to implement, while components in red are either mostly un-implemented or contain only placeholder code. The green "Current Best Move" is a continuously updated value and not a module per se.



Game State

The game state component is the literal representation of the state of the game at any given point. This component also contains most of the logic for gameplay, including move resolution and the data structures representing the game state.

The game state representation and game logical implementation is robust, clean and was non-trivial to implement, but is not the focus of the paper. The relevant code is found in the "representation" and "state" source packages.

Game Search

The game search component, on a high level, is responsible for taking in the current game state, all the input from chat and past experience, and deciding what orders to produce for a turn.

The goal was that the game search logic would remain independent of the game logic. All decisions on when to prune a search or about state utility would instead be generated by the heuristics package.

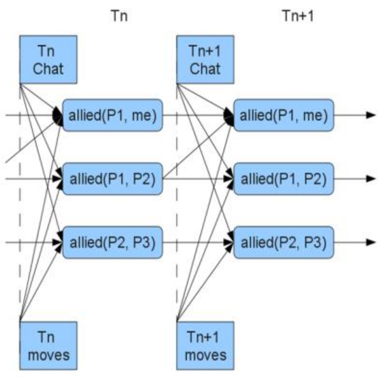
Heuristics

The objective of the heuristics package is to create an easy place to place all diplomacy-specific decision making. For example, the utility assigned to a diplomacy game state is generated with game-specific knowledge, and fits in the heuristics package. Likewise, the code which will take a game state and generate all reasonable moves for a given player also fits in here.

Dynamic Bayesian Network Belief States

The idea behind the DBN belief state was that it would provide, in short "the probability of anything being true" by aggregating the historical actions of players with the current state and their incoming chat. We wanted to use Dynamic Bayesian Network algorithms to keep this network updated.

Because we didn't have time to fully implement the chat and other diplomatic aspects of the game (which was the main motivation for this module), we were not able to implement this.



Diplomizer

Because of time constraints we were only able to implement the most basic press in bot—a subset of level 10 press, negotiations and alliances. This allowed the bot to respond and accept simple peace proposals from other bots. Upon receiving a proposed peace agreement Victorybot currently always accepts the peace offer. It then sets the corresponding field in the allies matrix in the diplomatic state to true. Then we send off our confirmation to our new ally to the server to be relayed to that ally.

Diplomatic State

The diplomatic state is intended to hold the state of the negotiations and agreements the Diplomizer makes Again, since the press/chat components weren't implemented, we were not able to implement this structure.

Design Decisions

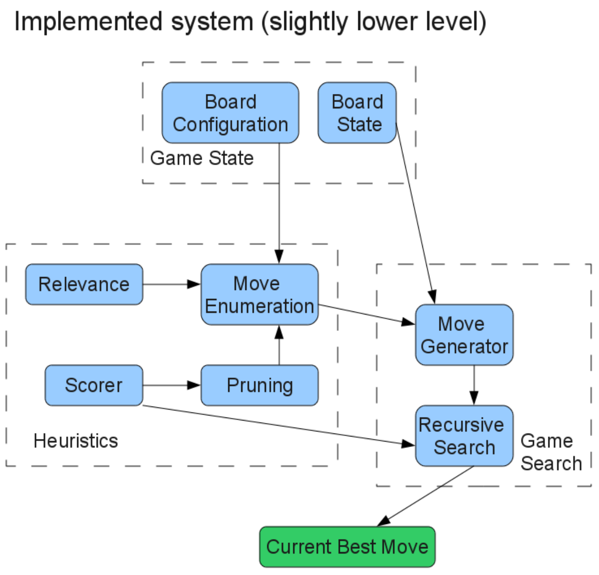
We decided that in the system we designed we would keep the search strategies generic, but keep the state representation domain-specific. While embedding the state representation in a formalism like STRIPS would have advantages, we felt it would have added a lot of complexity in implementation which we did not necessarily have time to debug.

On the other hand, we wanted to keep the search strategy as generic as possible. None of the game logic or heuristics are stored as a part of the game search package.

In general, the interfaces defining the behavior of modules of our code are generic and reusable, but their implementations are often game specific (especially for code in the heuristics and game state.)

Detailed Description of Implemented System

The figure below shows a more detailed diagram of our system as-implemented. This section discusses the details of each component and each of the implementations tested for each.



Game State

The game state component for ease of development was divided into two parts; one is the static *Board Configuration,* which holds information which is static throughout the game, such as territories and borders between territories. This package also holds most of the diplomacy-specific move resolution logic.

The *Board State* structure is an instance of a specific game state. After generating a set of orders for all players, the new game state is generated with:

newState = boardConfiguration.update(currentState, orders)

Heuristics

The heuristics package has been broken into four components, to allow different combinations of heuristics implementations to be tested.

Scorer

The scorer heuristic is responsible for taking a Board State and giving it a utility for a particular player. There is nothing particularly interesting here from an AI perspective, but is instead a collection of heuristics we generated from game-play experience. *NaiveScorer* implements this code.

Relevance

The relevance heuristic takes a board state and a player, and returns a set of players which are currently "relevant" from a tactical perspective—can their moves affect you on the next turn. The implemented code is in *NaiveRelevance*

MoveEnumeration

MoveEnumeration takes a board state and a player and generates a large number of potential viable moves for each player. There are a number of ways to implement this code; our code as implemented randomly generates move orders, and uses the remaining units to support moves or holds among the remaining units, making sure that duplicate orders are not added to the set. This code is in *NaiveMoveEnumeration*

Pruning

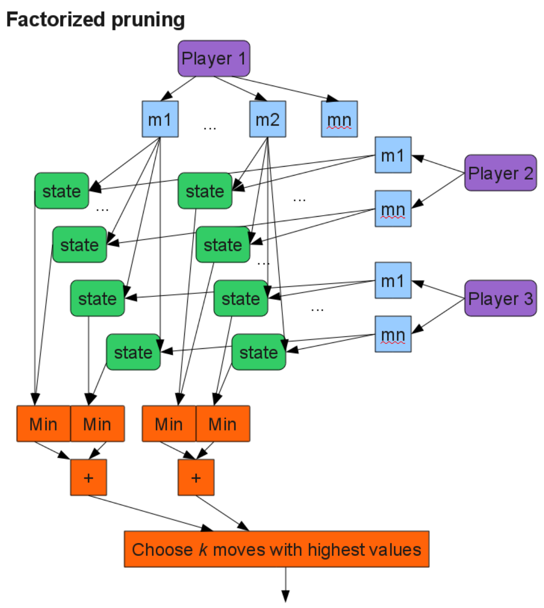
The goal of the pruner is to take a set of moves for a player and return only ones that are of at least somewhat high utility.

Our first implementation of this code is found in *NaivePruner*. The heuristic takes a move and looks at the outcome of the move assuming that no other player moves that turn.

As we tested our pruning code in action, we realized a weakness in our approach: in a situation where the moves of all 7 players enumerated to do even level 1 minimax search, we could tractably look at no more than 5 moves per player (5^7 = 78125). If we were only enumerating 5 moves per player, we needed to choose those 5 moves more carefully.

We developed the *FactorizedPruner* to help remedy this. The factorized pruner looks at the utility of each move by evaluating its utility against each other player, assuming that every other player submits hold orders. A min is done across all the moves for that player. Then, the quality for the move across all players is summed. The pruner will then select k highest-scoring moves as the pruned set.

The intuition is that if a move performs well against each player individually, it will likely fare well against all of them in conjunction, unless they are conspiring against you. The figure below attempts to illustrate this procedure.

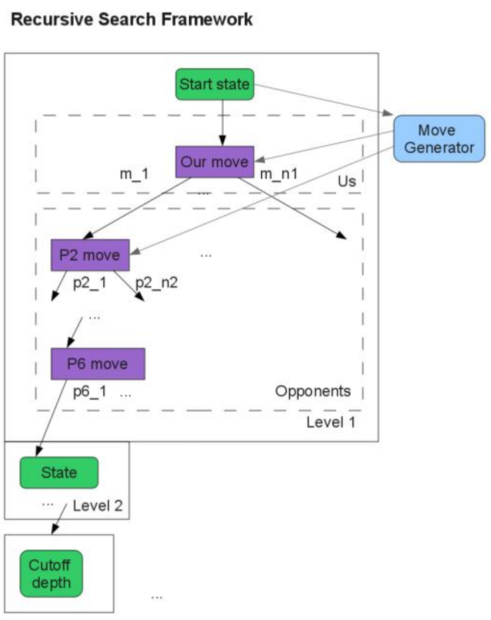


Game Search

The game search module is the heart of the AI implementation. The game search is at core a recursive traversal of the search space in mini-max fashion; however, the search is complicated by the fact that all moves in diplomacy are simultaneous; so instead of each level in the mini-max tree representing a new game state, only after a node for *each player* has been expanded will the new state be generated—in the case of diplomacy, this is 7 layers deep in the tree.

The figure below attempts to illustrate this. For each player node in the tree, our move generator component generates a set of likely moves for the player, which is then expanded.

On top of this, there are several ways that mini-max search can be extended to a multiplayer environment. The two we developed and discuss here we call MiniMax and ExpectiMax.



MiniMax

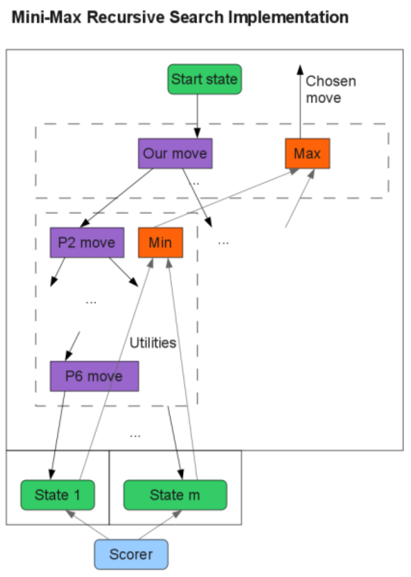
This search is a naïve extension of the 2-player minimax search procedure to a multiplayer environment. The algorithm takes a *paranoid* approach—the bot assumes that everyone is conspiring against it, and reacts to minimize that damage. At each min level, the worst possible outcome is passed up to the max level.

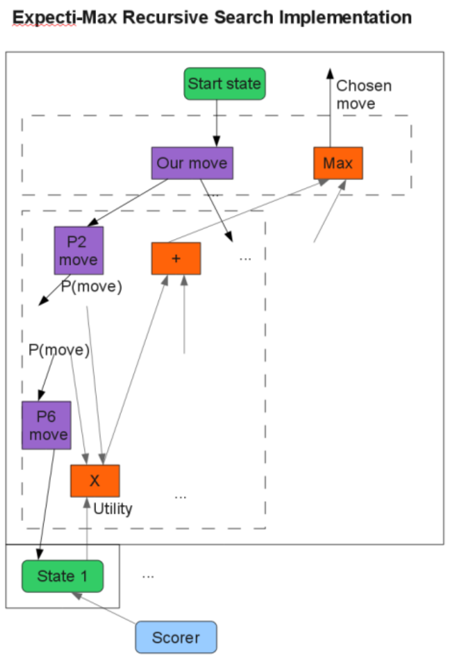
An illustration is shown below. This approach of course leaves much to be desired, so we decided to next look at a probabilistic approach.

ExpectiMax

The idea behind the ExpectiMax algorithm is that the utility of an outcome state should instead be weighted by the probability of it happening. The expected utility given a move for a player is the sum over all outcomes for that move.

The probability of an outcome state happening is the product of the probability of each opponent choosing the set of moves that leads to it—the probability of each arc from the origin state to the outcome state, as shown in the figure below. For now, we very approximately calculate these probabilities as a uniform distribution over the arcs. The eventual objective is that the belief state and diplomatic state will inform these probabilities.





Experimental Design

Despite our best efforts, our bot is not competitive with the existing bots we described in the related work section. In general the existing bots have sophisticated move generation heuristics which we did not have time to develop.

However, we still thought it would be useful to at least get performance results among implementations of our bot. In particular, we had two main design choices we wanted to compare the performance between: we wanted to compare the performance of our MiniMax and ExpectiMax algorithms, and we wanted to compare the performance of our Naïve pruning heuristic with our Factorized pruning algorithm. This gives us four configurations to compare: MiniMax search, naïve pruning; ExpectiMax search, naïve pruning; MiniMax search, factorized pruning; ExpectiMax search, factorized pruning.

The DAIDE environment was used in all the evaluations here. To try to get an unbiased evaluation of the bots, for each game, each of the 7 game slots was assigned a random configuration. We wanted to measure the following:

- Overall victory percentages; how many games did each configuration win?
- For each configuration, what is the probability that a particular bot of that configuration would be eliminated from the game?

Although we did not have the time to compute other metrics here, one future metric we would like to calculate is a cross-correlation among the strategies, to see which is most effective against each other configuration.

Experimental Results

The table below shows the raw results of the tests: for each configuration, how many games did it appear in, how many did it win, and in how many was it eliminated.

	Appear	Wins	Elim
ExpectiFactored	24	5	10
MinMaxFactored	22	2	9
ExpectiNaive	17	3	7
MinMaxNaive	14	1	9

Because we have a reasonably small sample size, and the games were randomly seeded, not all bots appeared in the same number of games. The table below at bottom shows how often each bot won and how often each bot was eliminated, as a percentage of the number of games it appeared in (note that the percentages do not sum to 100 because each configuration usually had multiple instances per game.)

We can see from the results that the configuration with Expected MiniMax search and factored pruning outperforms the other configurations in win percentages; in fact, in all cases, factored pruning outperforms our naïve pruning, and expecti-max search outperforms mini-max search. We can see that in this metric, MinMaxNaive markedly worse than other configurations. We were happy to see that the two new strategies we developed during the project (ExpectiMax and factorized pruning) outperformed the more naïve strategies.

	% Win	% Eliminated
ExpectiFactored	20.8	41.7
MinMaxFactored	9.1	40.9
ExpectiNaive	17.6	41.1
MinMaxNaive	7.1	64.2

General Discussion

Original objectives

The original goal of this project was to investigate how well researched search strategies worked in the context of the game Diplomacy. We ran into several large hurdles along the way and were forced to step back and evaluate different methods to achieve our goals of deep search that built reasonable sets of moves.

Another ancillary goal was for the bot to be able to compete well against other bots in the literature. However, we spent the majority of our development time implementing the DAIDE framework and developing our search strategies, and not as much tweaking our heuristics. So although Victorybot was generally competitive for a time against other similar bots, it was unable to win in the end. With improvements to the heuristic, we believe that Victorybot could win against existing Bots.

Obstacles

While developing our bot, we ran into a number of obstacles we did not anticipate. First, exhaustive move generation for more than just a few units quickly became intractable. Instead of spending time refining our game search, we had to spend time more intelligently generating our original move sets. This lead to our factorized approach, where we plan our moves one player at a time and then after doing that for each player we look at the best of those to decide on a move set. This allowed us to greatly increase our move generation set, while also increasing the speed of that generation.

We developed the expecti-max search in an attempt to compensate for excessive paranoia and over-conservatism by the bot. By evaluating the utility of a move not just by it's the worst-case outcome, but weighting the outcomes by their probability of occurring, the bot was able to perform much better, as evidenced by our (albeit preliminary) results.

Future Work

The main feature not completed was the Diplomizer; we originally planned for the Diplomizer to be a major component of our architecture, but were only able to implement a skeleton structure for it. Because we did not get input data from the Diplomizer, the DBN and Diplomatic state components could not be finished either. Our ultimate objective is to generate full diplomatic interactions, where the bot could rely less on its tactical acumen and more on clever negotiation and treachery.

The other main weakness of our bot currently is that the heuristics we developed did not take full advantage of ideas known to perform well in previous literature. We would like to use the move generation heuristic from DumbBot in conjunction with our game search.

One last idea for we had to improve heuristic performance was to train a neural network or decision tree classifier to identify high-quality and low-quality board states, instead of coding heuristics by hand. This would allow a domain expert to use qualitative knowledge that is hard to programmatically add.

Concluding Remarks

Overally we were happy with the progress we made and the insights found into how traditional search strategies can be extended to a game like Diplomacy. Although we did not end updefeating all comers, we think we were limited not by our overall approach but by the amount of time and effort that was (not) spent developing good heuristics.

We would like to give many thanks to the DAIDE community for providing the server and standardized protocol which allowed us to compare VictoryBot to the existing body of bots. Also thanks to Henrik Bylund for the communication and socket code we used as the base for our communications module.

References

Avalon Hill. (n.d.). *Diplomacy Rulebook Archive*. Retrieved 12 8, 2010, from Diplomacy Archive: http://www.diplom.org/~diparch/diplomacy_rules.htm

Haard, F. (2004). *Multi Agent Diplomacy.*

Hal, J. v. (n.d.). Retrieved 12 8, 2010, from Diplomacy AI: http://sites.google.com/site/diplomacyai/

Newbury, J. (n.d.). Retrieved 12 8, 2010, from Diplomacy : http://johnnewbury.co.cc/diplomacy/blabbot/index.htm

van den Herik, H. e. (2006). Current Challenges in Multi-player Game Search. *Lecture Note in Computer Science* , pp. 285-300.

Webb, A., & et, a. (2008). *Automated Negotiation In The Game Of Diplomacy.*