Building Werewolf AI Game Agent

Wangfan Li, Griffin Milas, Kerry Ngan

Introduction

Motivation

Werewolf is a social deception game similar to party games like Mafia. In this hidden role game, players are secretly on one of two teams: Town, or Werewolf. Town's goal is to discover the identity and vote out the Werewolf team, while the Werewolves' goal is to eliminate all members of Town without getting caught. Over a series of rounds, players will use role-specific abilities to gather information or attack other players, and then attempt to vote out players based on public discussion.

(*The full rules of Werewolf are described at length in the Appendix)*

What makes this game interesting from an AI's perspective is that there is imperfect information, variable roles, and the ability for all players to lie or tell the truth at any point. This makes Werewolf a very different game from many of the other perfect-information state scenarios we've discussed in class. In addition, although we have discussed ways to deal with non-perfect information in AI previously, we have not thoroughly discussed the concepts of asymmetric powers/abilities, information sharing, opponent modeling, or believability / deception, all of which make Werewolf an interesting and intriguing problem to tackle.

Challenges

One of the potential challenges in creating a Werewolf AI is encoding the concept of deception or trust, where the AI will need to learn to have a general sense of not trusting

every statement from agents. In Werewolf, it's not always best to take everything at face value, so having some idea of sorting through the publicly available information will be crucial to creating a powerful AI.

Another factor is figuring out how to actually code the AI. We ultimately were able to find a library called AIWolf that allowed us to implement an agent relatively easily. However, the competition itself is based in Japan. As such, some of the resources, instructions, etc. on the AIWolf website were in Japanese, making them inaccessible to us. In addition, the 'recommended' approach for creating AIWolf agents is to use Java, but we instead chose to use the Python version (because it was a language we were all more comfortable with).

Evaluation of our agent is also a difficult factor. In an asymmetric game like Werewolf, the winrate of each team is not guaranteed to be 50/50. Especially in a game where all the players are independent agents that behave, act, and 'learn' differently, having some way of gauging the relative strength of our agent as compared to a 'baseline' may be difficult due to the medium of Werewolf as a game.

The most difficult factor stems from Werewolf as a hidden role and a social interaction game. There is no revelation for any roles until the terminal state, and opposing parties disrupt each other's belief so that they may not find their solution. At its foundation, Werewolf is a game that challenges the user's theory of mind. Creating an agent for an online version of Werewolf limits the resources that a human can generally use to discern mental states such as hearing tones, facial expression, and other ways that humans habitually express information. If we request information from a human, they will be tempted to respond or be marked as suspicious, but if we request information from an agent and do not receive a response, they may simply not be programmed to do so. We are working with only the text that other agents give us and evaluating it based on the agents pattern or what other agents tell us. All these factors create a difficult task to solve.

Results Overview

Overall, despite these challenges, we were able to create an agent that plays Werewolf competently at a variety of roles. Utilizing Monte Carlo Tree Search and a simple evaluation function, we were able to create a bot that can simulate future gamestates and pick actions according to its beliefs. Our bot achieved an overall 47% win rate against sample agents and an overall 28% win rate against a team of Takeda agents, one of the AIWolf competition winners from the 2020 competition.

Related Work

Our project is based on the AIWolf competition that has run annually since 2019. AIWolf has 2 distinct branches that people compete in: NLP agents and protocol agents. Our project will be focusing on the latter, where agents communicate through a prescribed language set out by the AIWolf framework. This will allow us to focus on the implementation of the AI without getting 'bogged down' by first having to decode what information is being told to us before we can utilize that information for our agent.

There has been much research into using various AI techniques for a variety of board and card games. Most similar to our work is a paper by Jack Reinhardt, "Competing in a Complex Hidden Role Game with Information Set Monte Carlo Tree Search". In this paper, a few AI Agents are designed to play another social deception game that is quite similar to Werewolf called Secret Hitler. The agents put out by the paper include a selfish rules-based agent, an information-set MCTS agent, and a random agent (who chooses actions randomly). The conclusion of the paper is that "the Single-Observer Information Set Monte Carlo Tree Search (SO-ISMCTS) algorithm fares no better than the Selfish agent in games with Random, Selfish, and SO-ISMCTS opponents", citing the lack of bluffing ability, as well as the agent's inability to determine the secret roles of other players as the algorithm's main downfalls. The paper also points out that direct player communication was not implemented, and is a possible source of future work.

We took inspiration from this paper in our implementation of MCTS, especially in respect to the idea of implementing (at least parts of) ISMCTS into our algorithm.

In addition, another paper, "Emergent bluffing and inference with Monte CarloTree Search" by Cowling, Whitehouse, and Powley, was also important to our team. In the paper, another popular social deception game, The Resistance, is played using an Information Set MCTS agent. However, the paper goes much beyond a simple ISMCTS implementation, and targets behavior of bluffing and inferring other players' roles in order to create a powerful agent. In this way, not only is the agent benefiting from the MCTS, but it also has the power of constructing a model of its opponent as it searches and plays. Additionally, another paper, "Integrating Opponent Models with Monte-Carlo Tree Search in Poker" by Ponsen, Gerritsen, Chaslot, also focused on opponent modeling while using MCTS, this time in order to play poker. The paper concludes that MCTS is enough to beat more simple rules-based agents, but opponent modeling is an important part of making a strong MCTS poker agent. These papers not only solidified our interest in implementing MCTS in our own algorithm, but it also exposed us to the idea of opponent modeling and inference in a social deception agent.

Approach

Overall, our approach is to create an agent that uses a Monte Carlo Tree Search (MCTS). Whenever a decision is to be made, we will roll out nodes for a given number of times and simulate those games. We can then choose the child gamestate with the best possible reward. For our approach, we chose to use 2 different approaches. One of which is influenced by a simple 'evaluation' function/belief state, which uses the action of each agent (talking during the day, voting patterns, actions, etc) and evaluates how likely any given player is a werewolf. The second approach randomizes the hidden roles equally and chooses one permutation.

Monte Carlo Tree Search

Both of our agents perform a tree search whenever we are required to make a decision. For example, if our agent is playing as a Villager, whenever the agent speaks or votes, they perform a Tree Search which will then rollout out nodes for a specific number of times and then choose the node with the best reward.

Our Monte Carlo Tree Search utilizes an information set, a set of determinization of the hidden roles. In terms of werewolf, this means that we pick one out of all the permutations of roles that each player can possibly be. Each node will carry an information set that describes each player's roles. The rollout phase works by first choosing a leaf node that is unexplored which is chosen based on an upper confidence bound, expanding it by assigning all of its next children, and then simulating a game until it is terminated. The resulting win/loss will be back-propagated as a reward and the node with the best reward will be chosen to be returned.

Evaluation Function/ Belief State

Although we wanted to explore more complex ideas of opponent modeling, our simple approach to opponent modeling was to create an opponent evaluation function. This function is used by the agent Player1 to create a determinization of the game state. At the beginning of each Werewolf game, we create a distribution of our beliefs, with each player having an equally likely chance of being a werewolf. Each time our agent takes in information, we update our evaluation function and do a rollout of the MCTS based on it. The evaluation function looks for specific things: votes against our agent, other players speculating or accusing us of being a werewolf, agents accusing another agent of being a werewolf, etc. These actions will change our beliefs accordingly, making certain players more or less likely to be a werewolf based on our beliefs. Although these values were hard-coded for our example, we had toyed with the idea of using a neural network to learn the weights for any given action.

The chance of determining any specific game state is based on our belief distribution of what we think is our current gamestate: essentially, the more we believe them to be a werewolf, the greater the chance they appear as a werewolf in a search. We then simulated the determinization from our evaluation. In playing out the series of particular actions, the tree assumes that whichever person we vote for will be voted out no matter other's opinion.

Random Playout/ Random Determinization

In player 2, we again implemented the monte carlo tree search but using a different approach. Instead of a belief distribution, we chose to randomize the determinization and the playout strategy. While there is a lot of information that can be taken from each player, is it difficult to discern lies from truth if an agent was intentionally telling one. Since 2 out of 5 players are on the werewolf team, the lies will significantly skew the belief distribution. Using such methods may allow us to take the action that has the highest potential win rate.

For this monte carlo tree approach, we randomly select a determinization from the players that are alive and fix the positions of the ones that are dead. During the expansion process, we return a permutation of all the possible children nodes. During the simulation, we randomly select a child from all the permutations at every step until the playout ends.

Our agent calls the choose function to return the node with the greatest reward during each of the decision events in the game. The villager returns this value. The seer similarly does the same but also divines this value. For the werewolf and the possessed, the rewards are inverted during the monte carlo tree process and the target is updated to the one that they want to kill.

Evaluation & Results

There are 4 distinct agents in the evaluation, Sample, Takeda, Player1. Player2. Sample agent is an agent that acts randomly. Every decision it makes is randomized. The second agent is Takeda, a finalist agent in the WerewolfAI competition, which uses the previous talk events to evaluate the roles of other players. The third is Player1, which uses a MCTS with a belief distribution to evaluate the werewolf. The fourth is Player2, which uses a MCTS which chooses equally from all permutations during the rollout process. To control the environment we ran 1000 games each round.

Our results are shown in the image below. The topmost row consists of the possible roles for the game. The leftmost column is the name of the agents that played. For any cell (i,j) below the roles and to the right of the agent name, it gives the ratio of wins to loses playing a certain role. The rightmost column is the total percentage win rate for the agent over 1000 games for all roles. For example, in our first simulation out of five included in Fig 1, our Player1 agent played 200 games in the role of possessed and won 68 times, 200 games in the role of seer and won 102 times, 398 games in the role of a villager and won 229 times, 202 games in the role of werewolf and won 73 times. This leads to an overall win rate of 0.472 averaging over all roles.

One aspect that could explain the apparent win rate of Player2 is the fact that villager A conjecture for our low win rate against takeda in fig.1 is that the difference in evaluation favors takeda. Player1 and Player2 were designed to vote against the player that they believe to have the highest chance of being the werewolf and while Takeda agent is heavily reliant on the information from the talk events. With 4 other Takeda to provide information along with our agents' straightforward talk events, it would be more probable for Takeda to discern the roles. In the game with 4 Takeda against the Sample agents, we can see that it has a higher win rate and that might be due to it’s random nature which is hard to get information from.

Overall our winrate for Player1 seems decent when compared to other agents, and given an evaluation function the tree searches for the best move at the time for a villager role. For Player2, our agent performs similarly with a slightly lower win rate across the board to Player1.

In fig. 2, we can further see that our agents performed better than Takeda overall when against 4 sample agents. It is likely because our agent is less reliant on the talk events, we are not affected by fake information as much as Takeda.
In fig. 3, we ran another 1000 games against Sample agent but limiting Player1, Player2, and Takeda to the Villager role and the results were similar to fig. 2.

	BODYGUARD	MEDIUM	POSSESSED	SEER	VILLAGER	WEREWOLF	Total
Sample1	0/0	0/0	25/202	70/194	209/404	1/200	0.305
takeda1	0/0	0/0	91/193	151/206	245/402	42/199	0.529
takeda2	0/0	0/0	89/200	155/204	264/396	75/200	0.583
takeda3	0/0	0/0	72/200	132/200	307/398	114/202	0.625
takeda4	0/0	0/0	68/205	147/196	285/400	113/199	0.613

	BODYGUARD	MEDIUM	POSSESSED	SEER	VILLAGER	WEREWOLF	Total
player1	0/0	0/0	23/200	92/200	66/398	1/202	0.282
takeda1	0/0	0/0	102/205	139/196	61/400	66/199	0.568
takeda2	0/0	0/0	82/193	138/206	81/402	93/199	0.594
takeda3	0/0	0/0	80/202	124/194	95/404	105/200	0.604
takeda4	0/0	0/0	77/200	143/204	69/396	99/200	0.588

	BODYGUARD	MEDIUM	POSSESSED	SEER	VILLAGER	WEREWOLF	Total
player2	0/0	0/0	23/200	91/200	157/398	0/202	0.271
takeda1	0/0	0/0	87/205	137/196	265/400	92/199	0.581
takeda2	0/0	0/0	89/193	147/206	277/402	100/199	0.613
takeda3	0/0	0/0	96/202	142/194	269/404	90/200	0.597
takeda4	0/0	0/0	78/200	110/204	286/396	91/200	0.565

Fig. 1: 4 Takeda bots vs. 1 Other Bot

	BODYGUARD	MEDIUM	POSSESSED	SEER	VILLAGER	WEREWOLF	Total
Sample1	0/0	0/0	98/200	129/204	195/396	74/200	0.496
Sample2	0/0	0/0	96/202	104/194	253/404	111/200	0.564
Sample3	0/0	0/0	78/193	125/206	221/402	108/199	0.532
Sample4	0/0	0/0	88/205	102/196	226/400	92/199	0.508
player2	0/0	0/0	88/200	92/200	209/398	63/202	0.452

	BODYGUARD	MEDIUM	POSSESSED	SEER	VILLAGER	WEREWOLF	Total
Sample1	0/0	0/0	80/200	126/204	243/396	97/200	0.546
Sample2	0/0	0/0	86/202	115/194	251/404	90/200	0.542
Sample3	0/0	0/0	86/193	132/206	241/402	87/199	0.546
Sample4	0/0	0/0	88/205	117/196	220/400	61/199	0.486
player1	0/0	0/0	68/200	102/200	229/398	73/202	0.472

	BODYGUARD	MEDIUM	POSSESSED	SEER	VILLAGER	WEREWOLF	Total
Sample1	0/0	0/0	75/205	122/196	241/400	115/199	0.553
Sample2	0/0	0/0	76/202	125/194	248/404	122/200	0.571
Sample3	0/0	0/0	81/200	128/200	227/398	99/202	0.535
Sample4	0/0	0/0	86/200	117/204	223/396	77/200	0.503
takeda1	0/0	0/0	105/193	85/206	215/402	10/199	0.415

Fig. 2: 4 Sample Bot vs 1 Other Bot

	BODYGUARD	MEDIUM	POSSESSED	SEER	VILLAGER	WEREWOLF	Total
Sample1	0/0	0/0	110/249	168/261	142/239	138/251	0.558
Sample2	0/0	0/0	117/252	135/238	148/262	104/248	0.504
Sample3	0/0	0/0	97/246	142/253	135/250	115/251	0.489
Sample4	0/0	0/0	114/253	117/248	137/249	81/250	0.449
player1	0/0	0/0	0/0	0/0	562/1000	0/0	0.562

	BODYGUARD	MEDIUM	POSSESSED	SEER	VILLAGER	WEREWOLF	Total
Sample1	0/0	0/0	129/249	134/261	126/239	85/251	0.474
Sample2	0/0	0/0	116/252	136/238	149/262	123/248	0.524
Sample3	0/0	0/0	98/246	134/253	137/250	124/251	0.493
Sample4	0/0	0/0	111/253	142/248	134/249	122/250	0.509
player2	0/0	0/0	0/0	0/0	546/1000	0/0	0.546

	BODYGUARD	MEDIUM	POSSESSED	SEER	VILLAGER	WEREWOLF	Total
Sample1	0/0	0/0	86/205	123/196	231/400	113/199	0.553
Sample2	0/0	0/0	87/202	121/194	234/404	111/200	0.553
Sample3	0/0	0/0	79/200	122/200	223/398	109/202	0.533
Sample4	0/0	0/0	79/200	109/204	220/396	91/200	0.499
takeda1	0/0	0/0	110/193	84/206	210/402	17/199	0.421

Fig. 3. 4 Sample Bots vs 1 Other Bot (ran again)

Conclusion and Future Work

In conclusion, we have created an agent who plays Werewolf competently at a variety of roles. Utilizing Monte Carlo Tree Search and a simple evaluation function, we were able to create a bot that can simulate future gamestates and pick actions according to its beliefs.

Regarding future work, the problem that our tree faces is two fold, in that we lack an opponent model and also the simplicity of the villager role doesn't lend well to complex AI behavior. Both could be somewhat solved by developing AI for more complex roles like werewolf or seer, so that should definitely be the next step.

Another direction we could use is multiple observer information set MCTS (Cowling, Powley, Whitehouse. 2012), which should be an improvement to the default MCTS for more complex behaviors required by the advanced role such as werewolf, seer or bodyguard, and it is definitely something we want to implement for the future.

We could also implement as many tree MCTS (Cowling, Powley, Whitehouse. 2015) to track different information sets for each player. This would allow us to create a more reactive agent which can do actions like bluffing. Traditionally, MCTS like the ones that are implemented for our agent, considers only the determinization in a single player's point of view. Having many trees can allow us to sample information that is known to be false.

Additionally, work can be done on the evaluation function of our agent as well. Currently, the evaluation function weights for actions are hard-coded, and we only have a few specific actions chosen to actually trigger a change in our beliefs. By utilizing a more advanced technique of opponent modeling, or perhaps using a neural network or other technique to 'learn' these weights for a variety of actions, the agent could become much stronger in the future.

Acknowledgements

This project was created by Wangfan Li, Griffin Milas, and Kerry Ngan. Wangfan and Kerry worked the most on the coding, while Griffin focused on higher-level implementation, finding related work, presentation / write up, and organization.

Primarily, we utilized the AIWolf project's code (*http://aiwolf.org/en/*) for setting up the game, server, agents, and its communication protocol. We also used the code from Claus Aranha at Tsukuba University (*https://github.com/caranha/aiwolf_agent*) to use the framework in Python instead of Java (though AIWolf is primarily written in Java, this approach has been de-facto accepted by the AIWolf competition as valid). Finally, for testing and evaluation purposes, we downloaded and ran various AIWolf competition agents from 2021 (http://aiwolf.org/en/archives/2404), such as Takeda.

We partially based the code of our Monte-Carlo Tree Search implementation on a github abstract implementation by user Luke Harold Miles. (*https://gist.github.com/qpwo/c538c6f73727e254fdc7fab81024f6e1*).

Reference

Çelikok, Mustafa Mert, Tomi Peltola, Pedram Daee, and Samuel Kaski. “Interactive AI with a Theory of Mind.” *CM CHI 2019 Workshop*, 2019. https://arxiv.org/pdf/1912.05284.pdf.

Cowling, Peter I., Daniel Whitehouse, and Edward J. Prowley. "Emergent Bluffing and Inference with Monte Carlo Tree Search." IEEE CIG. September 2, 2015. http://orangehelicopter.com/academic/papers/cig15.pdf.

Cowling, Peter & Powley, Edward & Whitehouse, Daniel. (2012). Information Set Monte Carlo Tree Search. IEEE Transactions on Computational Intelligence and Ai in Games. 4. 120-143. 10.1109/TCIAIG.2012.2200894.https://www.researchgate.net/publication/254060888_Information_Set_Monte_Carlo_Tree_Search

Ponsen, Marc, Geert Gerritsen, and Guillaume Chaslot. "Integrating Opponent Models with Monte-Carlo Tree Search in Poker." 2010. https://www.aaai.org/ocs/index.php/WS/AAAIW10/paper/viewFile/1984/2462.

Reinhardt, Jack. "Competing in a Complex Hidden Role Game with Information Set Monte Carlo Tree Search." ArXiv.org. May 14, 2020. https://arxiv.org/abs/2005.07156.

Van der Kleij, A.A.J. “Monte Carlo Tree Search and Opponent Modeling through Player Clustering in No-Limit Texas Hold’Em Poker.” Thesis, University of Groningen, 2010. https://www.ai.rug.nl/~mwiering/Tom_van_der_Kleij_Thesis.pdf.

Whitehouse, Daniel. “Monte Carlo Tree Search for Games with Hidden Information and Uncertainty.” University of York, August 2014. https://core.ac.uk/download/pdf/30267707.pdf.

Appendix

Gameplay Rules of Werewolf

(*Gameplay of the AIWolf competition version of Werewolf varies slightly from the 'real' Werewolf game, so what is described below is the rules played by AIWolf)*

Setup

At the beginning of the game, players are distributed roles, which determines which team each player is on. Werewolf can be played with either 5 or 15 players, and the distribution of roles is based upon the number of players in the game, as described below:

Roles for a 5 Player Game

- 2 Villagers
- 1 Seer
- 1 Werewolf
- 1 Possessed

Roles for a 15 Player Game

- 8 Villagers
- 1 Seer
- 1 Medium
- 1 Bodyguard
- 3 Werewolves
- 1 Possessed

Objective

The objective of the game is to eliminate all players from the opposing team. More specifically, Team Town wants to eliminate all Werewolves, and Team Werewolf wants to eliminate all members of Team Town. Notably, this means that for Team Town to win, they need not eliminate

Possessed players. The game ends immediately when one of these two objectives are achieved.

Phases of Play

Werewolf is broken into two main phases: Day and Night. During the day phase, players discuss information publicly and have the chance to vote a player out of the game. After this, the night phase occurs, where players may preform special abilities according to their roles. After Night, a new round begins on the Day phase.

(Notably, the first turn is an exception to the normal turn order: because there is nothing to 'discuss' until abilities are used / players are attacked, the Day phase of the first round is skipped, and players proceed directly to the Night phase.)

Day Phase

During the Day phase, players first awake and are alerted if any players were eliminated during the night.

Talking Phase

Players are allowed to communicate information with each other. In each of 20 turns, all agents are allowed to send one message (as specified by the AIWolf framework), and each player can send 10 messages total each Talking Phase. Each turn, the order in which players speak is determined randomly.

Voting Phase

Players can now vote on one player to eliminate from the game. If one player receives the majority of votes, that player is eliminated from the game. In the result of a tie, a single 'revote' is performed. If the result is still tied, a random player is chosen among tied players, and that player is eliminated.

Night Phase

During the night phase, players can use their abilities according to their roles. Some players have no abilities, in which case they do nothing during the Night.

Roles and Abilities

The breakdown of what roles are distributed at the beginning of each game is determined by the player count (see *Setup*). Below are the descriptions on each role, its abilities, and any special rules.

Bodyguard

Team: Town
Ability: *Guard*

During the Night phase, bodyguards can choose another player to Guard. If that player is attacked by a Werewolf during the night, the attack is blocked.

Medium

Team: Town
Ability: *Werewolf Determination*

When a player is eliminated by voting during the Day Phase, the Medium is informed whether the player was a Werewolf or not (DOES NOT detect Possessed as a Werewolf).

Possessed

Team: Werewolf
Ability: None

The Possessed appears human to the Medium and Seer. Possessed cannot attack. Their objective is the same as the Werewolves (eliminate all Town members).

Seer

Team: Town
Ability: *Divine*

During the Night phase, the Seer can choose one player to Divine. The seer will know if the player is a Werewolf or not (DOES NOT detect Possessed as a Werewolf).

Villager

Team: Town
Ability: None

Werewolf

Team: Werewolf
Abilities: *Whisper, Attack*

Whisper: During the Night, Werewolves can whisper to one another. This chat functions and is structured similarly to the public chat, except that only Werewolves may see and communicate in it. It is mainly used to discuss information about bluffing roles and/or which member to Attack.

Attack: During the Night phase, Werewolves vote on one player to attack during the night. This vote is performed similarly to the vote that occurs in the Voting Phase.