Multi Agent Implementation Of The Werewolf Game

Davide Sangiorgi
davide.sangiorgi3@studio.unibo.it

Matteo Conti
matteo.conti16@studio.unibo.it

June 2023

The goal of this project is to develop a Multi Agent implementation of the famous game called Werewolf. This game is based on the rivalry between two factions of agents (i.e. farmers and wolves) who challenge each other during the development of each game in order to eliminate as many agents as possible from the opposing faction. This game contains multiple components common to multi-agent systems: there is an advanced communication mechanism associated with each faction, which also contains different privacy policies for each of these. Furthermore, each agent is independent and simultaneously implements both the principle of cooperation with respect to the agents of his faction and the principle of competition with respect to the agents of the opposing faction. In addition to this, we have also implemented several elements of this game in the form of "artifacts", mainly represented by the "perks" available to each agent. Finally, each agent has been provided with an "intelligence" mechanism through the structuring of a Bayesian network capable of representing the probabilities associated with each game event throughout its continuation. The results presented were obtained from the simulation of multiple games based, in turn, on multiple different configurations relating both to the game and to the strategies associated with each agent. The results themselves highlighted the correctness of the completed implementation, highlighted the differences with respect to the possible game policies associated with each agent and, at the same time, brought to light the effects of the artifacts with respect to the game strategy of each agent.

1 Requirements

The aim of this project is to develop and implement a multi-agent system with the ability to play the Werewolf game. In order to achieve victory, the agents must effectively communicate and collaborate with each other, while also being proficient in the use of artifacts.

The successful implementation of such agents would require them to possess the ability to navigate complex social interactions and understand the strategic implications of artifact use. The outcome of this project could provide valuable insights into the development of intelligent agents capable of operating in dynamic, social environments.

1.1 Game Rules

1.1.1 Roles and Goals

The Werewolf game is a social deduction game that revolves around two main roles: the Farmer and the Werewolf. Each player is assigned a role at the beginning of the game, dividing them into two opposing factions that compete for victory.

The Farmers aim to identify and eliminate the Werewolves in their midst, using their deduction skills to analyze the behavior of other players and spot any suspicious activity that could indicate a Werewolf's presence. Cooperation among Farmers is crucial to successfully eliminate players they believe are Werewolves.

On the other hand, the Werewolves' goal is to eliminate the Farmers and take over the village. To achieve this, they must develop group strategies to eliminate most dangerous Farmers from the game while at the same time avoiding detection by the Farmers.

Despite both competing for victory, Farmers and Werewolves must adopt completely different strategies. The Farmers are the majority at the beginning of the game, but they are not aware of the other players role and must use their deductive skills to identify the Werewolves and work together to eliminate them. The Werewolves begin as the minority, but they have specific moments in the game where they can communicate without the farmers' knowledge and they are aware of each player's role.

1.1.2 Game Phases

The game is divided into two distinct phases, namely the night and the day.

At night, werewolves communicate secretly to discuss their strategy and decide which farmer to eliminate. Additionally, during the night phase, some "perks" can be activated by the farmers. Once the werewolves have cast their vote and the farmers have utilized their "perks", the farmer who was voted upon by the werewolves is removed from the game.

During the day phase all players can discuss and the farmers should analyze other players behavior to detect werewolves. Terminated the discussion, all players cast their vote, the player who receives the majority is eliminated from the game and then the day phase ends.

1.1.3 Perks

Depending on the version of the game, at the beginning, some special abilities or "perks" are randomly assigned to a subset of Farmers. It should be noted that each Farmer is limited to possessing only one "perk" at a time. The subsequent listing will outline some of the most prevalent "perks" along with their respective descriptions:

- **Seer**: during each night phase, the *seer* has the ability to ascertain the role of a single player within the game.
- **Bodyguard**: at the onset of the night phase the *bodyguard* selects one player to protect. If that player if voted during night by werewolves, he/she will not be eliminated from the game.
- **Gunner**: the *gunner* is in possession of a certain number of bullets and can eliminate a player at a time during the day phase by shooting him/her, consuming one bullet.
- **Assassin** the *assassin* can target another player, and can change their selection during the day phase. If the Assassin is eliminated, the target is also eliminated.

1.1.4 Game Flow

The game starts with the random assignment of roles and "perks". Once the factions are defined, the starting game phase is the night, where the farmers are excluded from the chat and only werewolves can communicate. If the game is played in person, the farmers must close their eyes, while if it is played on the computer, the farmers simply cannot access either the chat or the voting. As anticipated, the discussion is followed by a voting phase in which the werewolves choose which farmer to kill. Before the end of the night all "perks" that can be employed during night are utilized. For instance, the *seer* may investigate a player's role, while the *bodyguard* selects the individual to protect. If there are no "perks" that can prevent the player chosen by the werewolves from being eliminated, the night concludes with the removal of the voted player.

The day phase follows and it starts with the discussion involving all players, including both farmers and werewolves. Werewolves aim to steer suspicion towards innocent farmers and avoid detection, while farmers must identify the werewolves among all players. Some "perks" like the *gunner* can be employed during the discussion, potentially affecting its course. The discussion is then followed by a voting phase, involving all players, which ultimately results in the elimination of the voted player and the end of the day. The end of the day determines the begin of the night, restarting the game loop.

All discussions and voting phases have usually a time limit.

1.1.5 Ending Conditions

Each faction's objective is to eradicate all players from the opposing side, resulting in the game concluding once one faction has been eliminated entirely. For werewolves,

obtaining a majority isn't sufficient, as the "perks" can unexpectedly disrupt the game's balance.

1.1.6 Special Cases

In this section we exploit some rules introduced to solve ad hoc situations.

Voting Deadlocks A deadlock may occur during voting, particularly in day phase, where no player receives the majority of votes. The game version determines the course of action, and there are two typical approaches. The first entails holding a new round of voting, but this time with only the players who received the highest number of votes being eligible for selection. The second option is to have no elimination as a result of the voting process.

Voting Turns The order of day voting is clockwise starting from the first player. However, to prevent the first player from being the same every day, a random player is selected for the first day's vote. From then on, the first player is chosen in a clockwise order.

Voting Restrictions The only restriction on voting is that you cannot vote for yourself.

2 Requirements Analysis

2.1 Adopted Game Rules

This section will detail the game version utilized for this project, elucidating our choices and the methods employed for handling exceptional situations.

Roles and Goals The core of the game is not changed, the competing roles are farmers and werewolves and their goal remains to eliminate each member of the opposite faction from the game.

Game Phases Even the game phases remain unchanged, with the alternation of night and day as discussed before.

Perks We selected a limited number of "perks", in particular the *seer* and the *bodyguard* that are defined before. The only notable specification is that neither the *seer* nor the *bodyguard* can target themselves when using their "perks".

Game Flow To clarify the game flow, we need to provide a few more details. There is no discussion among the werewolves during the night phase; instead, they immediately cast their votes, while the farmers are asked to use their "perks", if they have any. The night concludes with the elimination stage, in which the werewolves' voted farmer is removed from the game, unless it is safeguarded by the *bodyguard*. During day the potential elimination of a player is announced to all players. In all cases, whenever a player is eliminated, its role is revealed. During the discussion phase, players may only make accusations about which player they believe is a werewolf. In this way, the accusing player anticipates what its vote might be. Another implication is that the *seer* is unable to expose itself, which doesn't deviate much from the norm as the *seer* typically refrains from doing so.

Ending Conditions Since the only "perk" that can interfere with a farmer elimination is the *bodyguard* and the *bodyguard* can not protect itself, the ending conditions can be revised: Farmers win only if all the werewolves are eliminated from the game, while Werewolves can win also if they reach the majority of the players. The werewolves win also if only one farmer and one werewolf are left in the game.

Special Cases The voting process is managed in a slightly distinct manner. Firstly, the order of voting holds no significance, and the previously mentioned precautions are disregarded. Secondly, the voting outcome is divided into four categories, namely: valid, not valid, deadlock, and not deadlock. An outcome is considered valid only when two or more players vote for the same target and it is deemed a deadlock if no single player garners a majority of votes. During day phase, the voting is repeated until it is no longer a deadlock (and, as consequence, it is neither not valid). Conversely, during the night,

a valid vote suffices, and if the outcome is a deadlock, no farmer will be eliminated. It is forbidden to vote for themselves or select themselves as the target of any "perk".

3 Design

3.1 Structure

In this section we investigate all the strategies used for modelling both entities and components of the WereWolf game following the Multi Agent System principles.

3.1.1 Entities

First of all we start describing entities mainly responsible for both action and control tasks carried out during the game execution.

Controller This is the most important entity of our entire scenario. This entity represent the common *nameserver* controller typically implemented in Multi Agent Systems in order to regulate and assess agents communication and interactions with both other agents and the environment itself.

In particular, this entity has the following goals:

1. it controls the entire application workflow: it starts/ends/pause the game, it tracks agents actions and messages and regulate the *nameserver* creation.

2. it defines the game flow: it starts and ends each phase (i.e. day, night) at turn and communicates, at the same time, all the information related to these changes directly to the agents.

3. it validates the agent behavior: it checks all the agents votes and accusation constantly ensuring that game rules are not violated.

4. it controls the agents lifecycle: it deliberates both the agent's registration to the *nameserver* and their deletion from both the game and the *namespace*.

5. it defines the "state" of all agents: it is in charge to both connecting agents after their initial registration (giving the agents the possibility to communicate with each other) and, at the same time, to mask all death agents actions and removing all their connections with other agents.

6. it guarantees the integrity of the application: in terms of security threats, this entity ensures that any compromise (hacking) of any entity other than itself does not compromise unintended effects on the rules or the functioning of the game.

Agents This entity follows the standard definition in *agents* in Multi Agent System environments [8]. These agents are developed as independent entities capable of both acting and communicating with each others.

In this project we have developed two group of agents (i.e. farmers and wolfs) differentiated by the role itself that they assume in the game. Each group of agents have the goal of winning the game.

Additionally to all the basic communications and acting features, we have also integrated inside each agent an additional component called *brain* which represent the source of intelligence exploited by each agent during the game.

It is important to remark that all communications and actions performed by these agents are supervised by the *controller*.

Artefacts In our applications *artifacts* are represented as perks capable of powering agents with extra-functionalities or giving access to them at hidden information that agents themselves can exploit in order to raise their winning probabilities. For instance, *seer* artefact allow agents to access to private information related to other agents. This access is always regulated by the main *controller* entity.

3.1.2 Components

Nameserver The *Nameserver* is entity in charge of the registration of the agents. This operation is carried out by saving agents uuids[10] inside an organized registry called *namespace*. This registry associates to each agent's uuid its *address* which corresponds to a pair of *protocol* and network *hostname:port* through which the *nameserver* is capable of redirecting physically all the messages to the "virtual" location of each agent.

We recall that in a distributed Multi Agent System agents can be deployed over different networks and location, which results also in exploiting different communication protocols and different deployment strategies (e.g. cloud, on-premise, ...).

The main goal of the *nameserver* is to abstract the physical protocol and driver needed to reach and communicate with an agent. Moreover, agents themselves can communicate with each other transparently, thanks to this entity.

Game The Game component represents and encapsulates all the information related to game during its execution.

In particular we developed three main components associated to this one:

1. *GameState* this component keeps track of the changes as a result of agents' actions (e.g. it knows which agents can vote, which agents are death, ...)

2. *GameHistory* this component is responsible for storing the complete record of all events that take place during the game, allowing us to perform statistical analysis on the games.

3. *Environment* this class includes a subset of the information associated to the game which can be shared with each agent (not all agents can know the roles of the other ones ...).

Brain This component represents one of the game changing integrations that we made in order to power agent capability with a base level of "intelligence". This is based on the representation of the agent's internal knowledge in terms of cause-effects probabilities

with the goal of anticipating the actions of other agents and build a strategy to fight them back.

Belief This component represents the internal "knowledge" acquired and developed by each agent during the game. These information are the foundation data used by the brain of each agent in order to develop the feature of "reasoning" and "training".

Messages This component represents the main data structure used by each agent to communicate with each other. In particular, we developed four kind of messages:

1. *DayDiscussionMsg* message used by agent to communicate during the day discussion phase
2. *VotingMsg* message used by each agent to share its vote with all the other agents
3. *WolfVotingMsg* message used by wolfs to communicate their vote with themselves
4. *AgentDeathMsg* message sent by each agent once it has been killed/removed

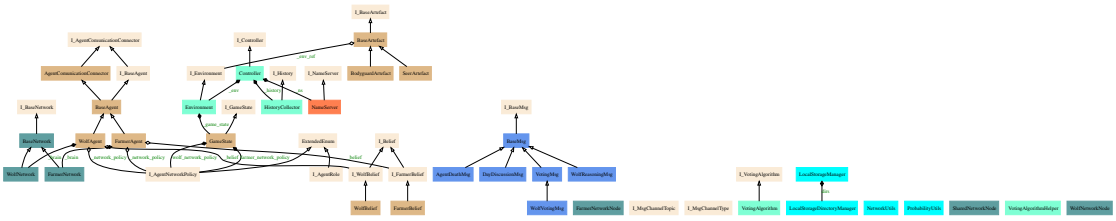

Figure 1: UML class diagram of the main entities and components developed.

3.1.3 Modularization

In this section we explain all the details regarding the modularization of main entities and components.

Nameserver The main modules implemented by this entity have been developed in order to carry out the following tasks:

1. *Communication* sets of low-level protocol to enable transparent agents communication
2. *Low-Level Agent's Lifecycle*: *namespace* registration, deletion, deployment location/information ...
3. *Agent's UUID Assignment* creation of agent's real uuid (i.e. masking) in order to hide the sensitive agent's sensitive deployment information

Controller The main modules implemented by this entity have been developed in order to carry out the following tasks:

1. *Game Representation*: game state, game history, environment, ...
2. *Agents Interactions*: regulates agents interaction and communications
3. *Roles Assignments*: in charge of assigning roles at the beginning of the game
4. *Roles ACL*: combined with *class encapsulation* regulates the game information access by each agent given their role
5. *Artifacts Assignments*: in charge of assigning artifacts at the beginning of the game
6. *Election ACL*: validate election results, asses "stalemate" occurrences

Agents The main modules implemented by this entity have been developed in order to carry out the following tasks:

1. *Brain Implementation*: store the probabilities distributions associated to each events for modelling the game dynamics
2. *Brain Training*: update knowledge about other agents as long as the game evolves
3. *Brain Inference*: exploit the hypothesis about the information regarding other agents in order to correctly develop a proper voting strategy
4. *Beliefs Update*: change its internal knowledge about the game in order to reflect the changes during the game evolution
5. *Artifacts Usage*: upon permission given by the controller, it exploit the artefact in order to power it capability of playing

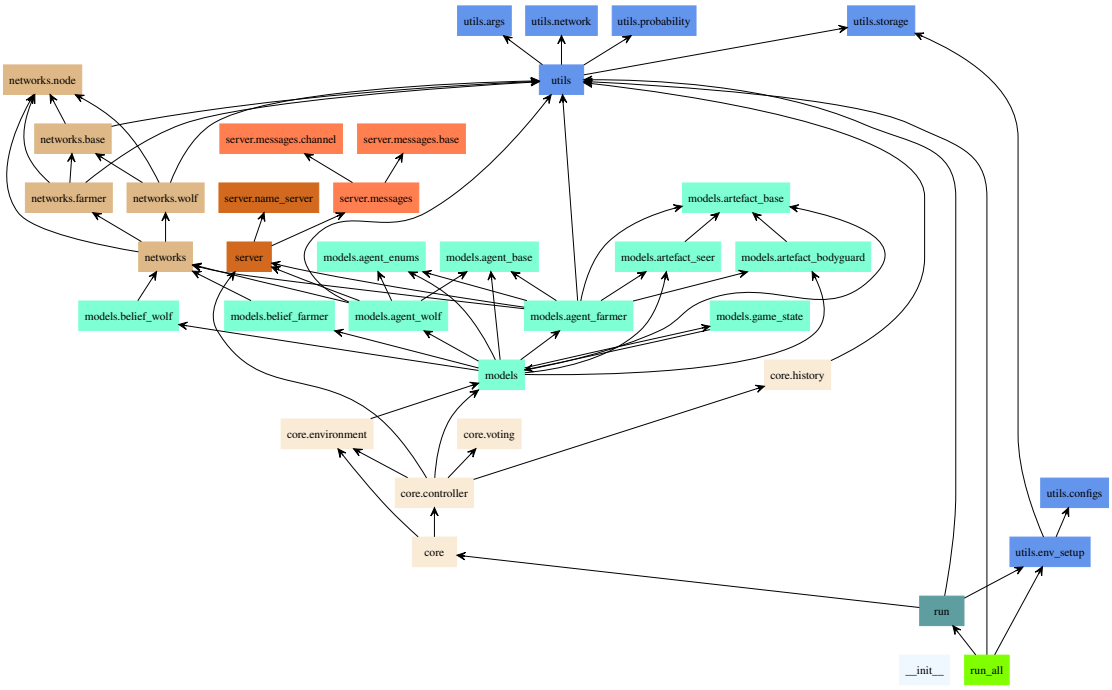

Figure 2: UML packages diagram.

3.2 Behavior

The base mechanism adopted for developing the behavior of each agent is based on one of the main computer programming principle adopted in the context of object oriented programming: this paradigm is called *class encapsulation*[6].

Agents

Beliefs Each agent has directly access to the *Environment* data structure and an integrated mechanism capable of constantly (i.e. at the end of each game phase) comparing its internal beliefs with information contained in this data structure. The final goal is to make the agents able to always stay up to date with the changes deriving from the triggering of different events (by other agents) within the game.

Brain Each agent can perform two groups of operations related to its encapsulated *brain*. The first group of operations includes the so called "standard" operations such as updating the "evidences/latent-variables" (i.e. notion related the probability modelling scope) of performing inference based on the probabilities themselves. The second group of allowed operations includes the so called "advanced" operation which include:

1. *Re-Training*: the agent can re-train its brain by updating each event probabilities modelled (see the 4.1.3 for more details) after the end of the day phase

2. *Initial Knowledge Loading*: the agent can preload pre-computed events probabilities (based on the current game configuration) modelling generic dynamics of the Werewolf game (see the 4.2.1 for more details).

These sensitive operations are regulated by the controller which assess the integration of these capabilities directly into the agent by checking the current global configuration selected from the user.

Status The *controller* entity regulates the "activeness" of an agent during the game. Once an agent it is selected to be killed (through an election process), then the controller remove this agent from the active agents list and then communicate to the killed agent it's new status. Once the agent received from the controller this information, then it send a message to all other agents in order to communicate its new status.

Game State Scope Each agent, based on its role, has access to a different set of information about the game and other agents. For instance, werewolves know, by construction, all other agents' roles, conversely to farmers which do not know anything more than their own role.

Controller The *controller* entity randomly assign both roles and artifacts randomly based on the current global configuration selected by the user. Moreover, there are other important tasks carried out by this entity: it also perform the election mechanism (see the 4.2.2 for more details) starting from the set of votes received by each agent. Although, at the same time, it applies logic for invalidate the election result in case of votes "stalemate" occurs (i.e. the two or more voted agents have the same number of votes).

Controller & Nameserver Since agents do not implement directly any kind of mechanism to communicate directly with other agents, each agent has natively implemented a mechanism for creating a communication channel through which it can share messages with the "world". Indeed, the *controller* share with each agent all the **publicly available** channels (i.e. wolfs also integrates a private communication channel with each other) of each agent, in order to allow reception of messages for anyone. The more important advantage of this approach relies on the fact that each agent can share a message with other agents, but it does not know which are the agent subscribed to its channels. At the same time, an agent can subscribe (through the mediation of the *controller*) to another agent channel, but it can only receive messages from this. This mechanism is based on a famous communication paradigm discussed in the 4.1.2.

Moreover, an additional level of security it is based on the fact that each type of message is serialized in an unique way. As a consequence, the *controller* at the beginning share with each agents only the message serialization policy needed for its role. In this

way we prevent the case in which if a compromised agent is able to access to private message (e.g. a farmer access to a wolf message), hence it can not de-serialize it since the *controller*, during the initialization phase, has not shared with it the de-serialization specs for this particular message.

4 Implementation Details

4.1 Main Components

4.1.1 Nameserver and MAS Controller

The Multi Agent System *nameserver* capability have been implemented through the use of the *osbrain* library.

osBrain is a general-purpose multi-agent system module written in Python and developed by OpenSistemas. Agents run independently as system processes and communicate with each other using message passing. osBrain uses ØMQ for efficient and flexible message passing between agents. It also uses Pyro4 to ease the configuration and deployment of complex systems. [1].

4.1.2 Communication

The low-level *communication* capabilities have been implemented through the use of the *Pyro4* library.
Pyro enables you to build applications in which objects can talk to each other over the network, with minimal programming effort. You can just use normal Python method calls to call objects on other machines. [3].

Pattern The reference communication pattern used for allowing agents to communicate with each other is called *Publish-Subscribe*.

In software architecture, publish–subscribe is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers, but instead categorize published messages into classes without knowledge of which subscribers, if any, there may be. Similarly, subscribers express interest in one or more classes and only receive messages that are of interest, without knowledge of which publishers, if any, there are. [9]

The main Multi Agent library [1] used to develop this project includes an implementation of this pattern thanks to behind the scenes integration of the already mentioned Pyro library [3].

As already mentioned in the Design section (3), this pattern allow us to preserve mutually exclusion of direction interactions between the agents.

4.1.3 Agent Brain

The *brain* capabilities have been implemented through the use of the *pgmpy* library.

pgmpy is a pure python implementation for Bayesian Networks with a focus on modularity and extensibility. Implementations of various algorithms for Structure Learning, Parameter Estimation, Approximate (Sampling Based) and Exact inference, and Causal Inference are available. [2]

These capabilities include the representation of the game events as Conditional Discrete Probability Distributions (cpds) [7], the update (aka. training) of this cpds and the inference process through which we extract probabilities of a given cause given the "evidence" probabilities of its effects.

Model The reference model used for modelling the structure of these probabilities it is called Bayesian Network [5].

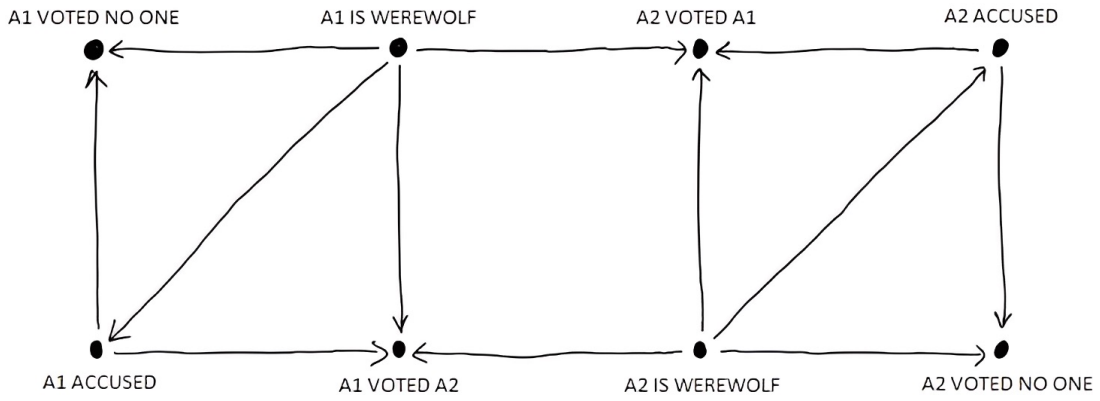

Figure 3: example of an farmer brain structure given a game with only two agents

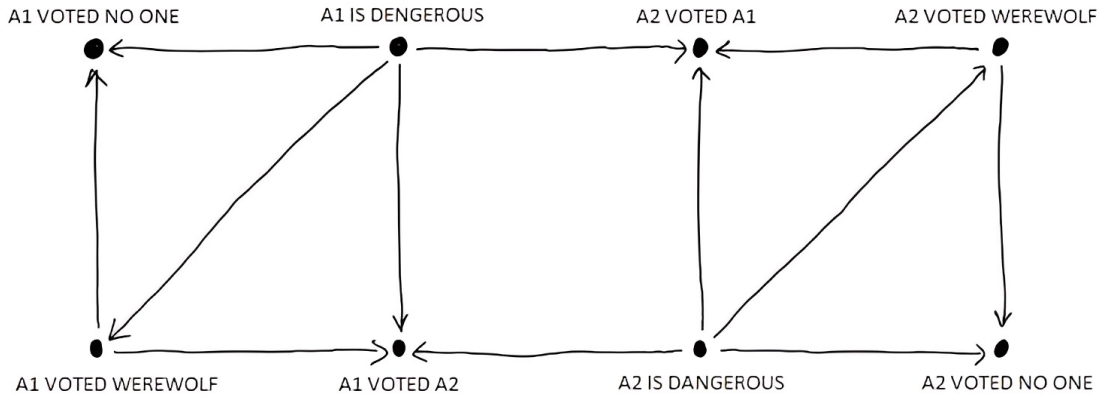

Figure 4: example of an wolf brain structure given a game with only two agents

Estimator The reference estimator used for carrying out the inference procedure based on "evidence" probabilities it is called Expectation Maximization [4]

Conditional Discrete Probability Distribution Each event probability was represented as (Tabular) Conditional Discrete Probability Distribution (cpds) [7].

4.2 Additional Components

4.2.1 Agent Initial Knwoledge Generation

The data structure containing the base cpds modelling all the combination of events occurrences of Werewolf game events has been computed separately through the simulation of all possible combinations of events values.

4.2.2 Voting Algorithms

The Voting algorithm is based on the principle of selecting the most voted agents. In order to do so, it collects all the votes from each agent and then it carries out an advanced extraction based on the resolution of possible stalemate cases (where possible).

4.2.3 Agent Voting Mechanism

Each agent queries its own brain (cpds) in order to extract the probability associated to the role of each other agents providing as evidence the information contained inside its own beliefs. Once he has done that, he models the inference results as a weighted probability distribution build in order to associate to each agents' uuids (which are the distribution entries) the related probability of its role as weighting factor (representing how much this agent is sure about the role of other agents).

5 Self-assessment / Validation

5.1 Test Driven

Unfortunately, due to time constraints, we have not been able to develop proper unit-tests.

Despite this, we list below all the functional and structural (i.e. source code) tests carried out during the development of this project. All the following tests have been made and form the basis which guarantees the integrity of the development and the correctness underlying the results obtained.

5.1.1 Structural

1. the application raise an exception in case the user create an invalid configuration for the game
2. reliability of the voting algorithm based on the sampling from a weighted distribution (i.e. if an agent has probability equal to 1 and all the other have probability zero, then the algorithm must choose always the former)
3. agent's message de-serialization in face of non-standard parsing of custom data structures
4. the application must ensure that the night phase will be played always before the day phase
5. the controller asses that compromised farmers cannot vote during the night phase

5.1.2 Functional

1. in case we have 9 farmers vs 1 wolf, then the farmers must always win (farmers are way more than wolfs)
2. in case we have a lot of agents, we have to play a sufficient number of times the day phase (this assessment is based on the assumption that the maximum number of agents that can be killed during an entire game cycle is known)
3. farmers powered with the *seer* artefact must vote always for a wolf (the only exception is when they have not discovered yet a wolf)
4. wolfs must always vote for a farmer by construction

6 Experiments

In our implementation, the werewolf game can be tackled by considering three primary factors: the farmer's policy, the werewolf's policy, and the perks. The policies employed by farmers and werewolves influence the decision-making and voting process of each player, while the perks add complexity to the game by providing advantages specifically for the farmer's faction, as previously explained.

In the subsequent sections, we conduct a thorough examination of the impact of the mentioned variables using graphs and statistics. We executed 8 games for each of the 412 configurations, each with a unique combination of the aforementioned parameters. Additionally, each game was played with varying numbers of farmers and werewolves. We recorded a comprehensive history of the game, documenting all actions performed by the agents.

For the upcoming comparisons, we filtered the previously mentioned histories based on the desired configuration for which we intend to compute our statistics. For example, we can select those histories where the farmer's policy is random, subsequently we compute the average statistics across all the filtered histories.

6.1 Metrics

Before presenting the graphs, let us provide a brief overview of the metrics computed for each set of configurations. Due to the complexity of the farmer faction compared to the werewolf one, considering factors such as perks and the absence of pre-training for werewolves, all computed metrics are focused exclusively on farmers.

First we present the **percentage of victories** achieved with the chosen set of configurations. This percentage is presented both as an absolute value and as a graph, related to the percentage of agents acting as farmers. This is because the proportion of the faction's population has the most significant impact on achieving a victory.

Next, we calculate the **correctness** of farmers' votes, which represents the proportion of farmers who correctly voted for a werewolf.

A more complex metric that we introduce with the purpose of measuring the collaboration between farmers is the vote **compactness**. This metric examines all possible combinations of two farmers' votes and determines the fraction of those combinations where both farmers vote for the same target. Essentially, it quantifies the extent to which farmers are able to act cohesively as group. Due to the large number of combinations involved, the denominator of this metric is typically high, resulting in relatively low values for vote compactness. While we do not have a definitive standard for determining what values should be considered high or low, we utilize this metric solely for comparisons among different configurations. As such, it still provides insights into the level of collaboration among farmers in each configuration.

Given that the *seer* perk is both crucial and vulnerable in the game, one of the most challenging objectives is typically to ensure the *seer* survives for the longest duration possible in order to maximize the benefits of this perk. Additionally, considering the possibility of having multiple *seers* in a game, we have chosen to depict the percentage of

seers who remain alive as the game progresses. To achieve this, we calculate the expected total number of *seers* for the entire set of configurations and continuously monitor the count of surviving *seers* at each turn of the game.

We have also generated a plot depicting the average lifespan of the *seers* in the analyzed configurations. This plot is presented in relation to the percentage of farmers in the game, aiming to determine if any specific configuration benefits more than others from a higher proportion of farmers.

6.2 UseCase 1 - Farmer Policy

In this section, we conduct a comparison between the Bayesian network policy, as described in the previous section, and a completely random policy. Another factor to consider regarding the farmer's policy is the use of pre-training, as explained earlier. We proceed to analyze three categories of configurations: those with the farmer's policy set to random, those with the Bayesian farmer's policy, and those with the pre-trained Bayesian farmer's policy. To simplify the presentation, the policy based on Bayesian networks will be referred to as the *network policy* in the following graphs.

Below, we provide the percentage of victories for each configuration:

Policy	**Victories (%)**
farmer random policy	37.03
farmer network policy without pre-training	46.34
farmer network policy with pre-training 1	45.15

The network policy clearly outperforms the random policy, while the pre-training did not effected much on the victories, resulting even slightly worse than the alternative without pre-training.

The resemblance in outcomes between the pre-trained and non-pre-trained network policies is further emphasized in the subsequent plot displayed in Figure 5. In this plot, the network policy options consistently outperform the random policy.

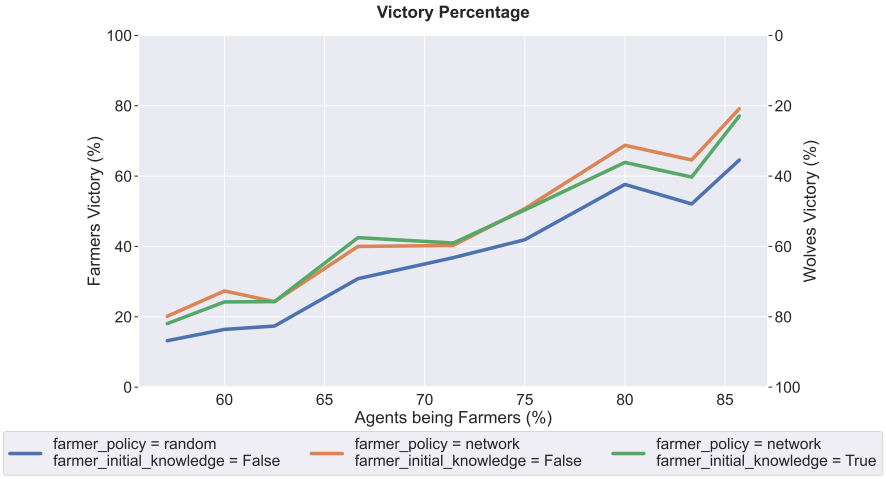

Figure 5: Percentage of victory related to the percentage of total agents in the game being farmers. We compare the average result across all games with farmer policy set to random, all games with farmer policy set to Bayesian network policy (network policy) and all the games with Bayesian network policy with pre-training.

Based on the observed flat trend in Figure 6, there seems to be minimal correlation between the percentage of farmers in the game and the accuracy of their votes. However, it is worth noting a slight negative slope in the plot, particularly with a significant drop of nearly 20% when the percentage of farmers exceeds 85%. This drop could be attributed to the inherent difficulty in detecting werewolves when their number is relatively low. The unusual value observed for the network farmer policy without pre-training warrants further investigation.

This particular configuration exhibits an unexplained additional drop in the right plot, specifically on day 3, which requires further scrutiny. Since the majority of games are won by werewolves (refer to the statistics in 6.2), the increasing accuracy displayed by farmers with a random policy is primarily due to the rising percentage of werewolves in the game. As the percentage of werewolves increases, the probability of farmers voting for a fellow farmer decreases. Nevertheless, there is an average improvement of nearly 10% in the early stages of the game when comparing the random policy to the network policy with pre-training.

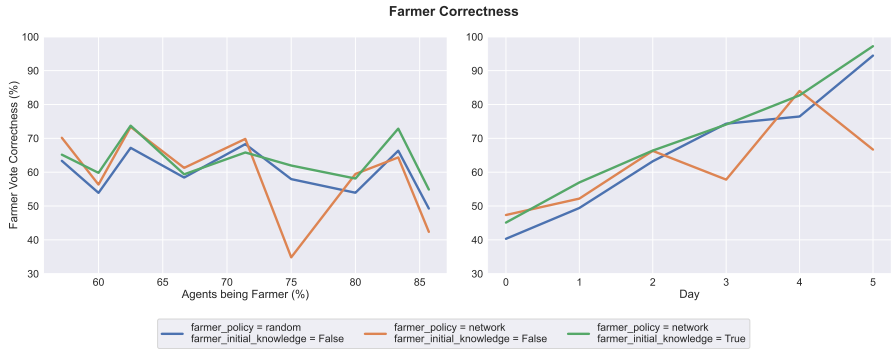

Figure 6: Percentage of farmers voting werewolves, on the left computed w.r.t. the percentage of farmers in the game, on the right w.r.t. the day of the game. We compare the average result across all games with farmer policy set to random, all games with farmer policy set to Bayesian network policy (network policy) and all the games with Bayesian network policy with pre-training.

The compactness of farmers (Figure 7) appears to be positively impacted by the implementation of farmers' policy, particularly when no pre-training is employed. However, the plot on the left suggests that the percentage of agents assuming the farmer role has a stronger influence on compactness compared to the specific farmers' policy. With regards to the plot on the right, the behavior of the farmer network policy without pre-training generally aligns with expectations, except for day 4. However, when pre-training is also applied, the advantages of the network policy diminish by day 2. As the game progresses, the lack of updates in the probabilities of the pre-trained network policy causes the aforementioned configurations to perform even worse than the random policy. Unfortunately, the implementation of incremental training was not possible due to limitations in the library. The significant drop observed on the 5th day merits further investigation, although it is likely influenced by noise. It is important to note that only a small percentage (4.32%) of these configurations reach the 5th day of the game, resulting in an average computed from a much smaller number of values.

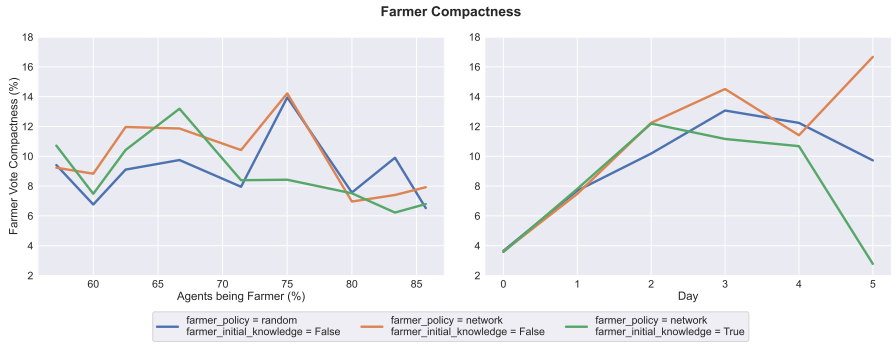

Figure 7: Percentage of farmers voting the same target, on the left computed w.r.t. the percentage of farmers in the game, on the right w.r.t. the day of the game. We compare the average result across all games with farmer policy set to random, all games with farmer policy set to Bayesian network policy (network policy) and all the games with Bayesian network policy with pre-training.

A highly surprising pattern emerges from the data presented in Figure 8, which illustrates the percentage of surviving *seers* throughout the duration of the matches. Contrary to our expectations, the plots representing the configurations with network policy do not consistently outperform the random reference. This is particularly unexpected given the previously observed positive impact, such as in the percentage of victories.

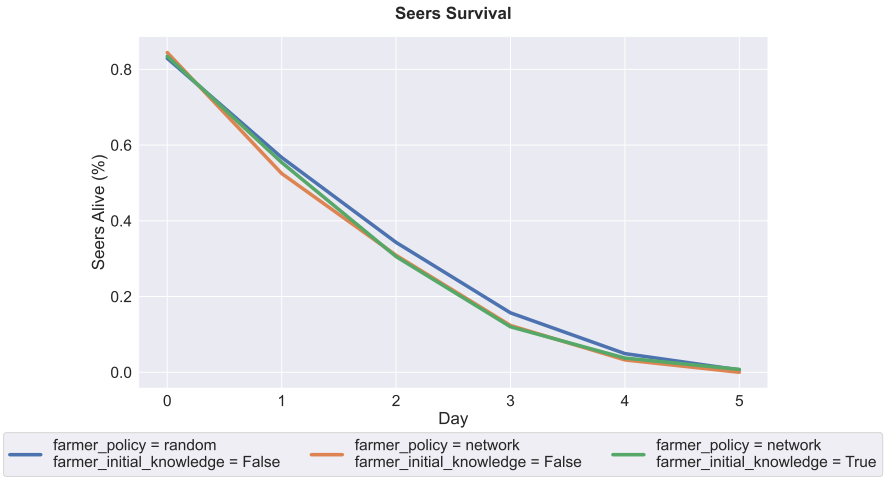

Figure 8: Average percentage of *seers* still alive at certain day of the game. We compare the average result across all games with farmer policy set to random, all games with farmer policy set to Bayesian network policy (network policy) and all the games with Bayesian network policy with pre-training.

Figure 9 reveals that the farmer policy does not appear to exert any influence on this particular type of plot.

Figure 9: Average number of turns *seers* survive, given the percentage of agents being farmers. We compare the average result across all games with farmer policy set to random, all games with farmer policy set to Bayesian network policy (network policy) and all the games with Bayesian network policy with pre-training.

6.3 UseCase 2 - Werewolf Policy

Similarly to the previous section we compare the Bayesian network policy with a completely random policy, but referring to werewolves. In this case we have only two set of configurations instead of the three we had for farmers because werewolves Bayesian network was not pre-trained. Like in previous section the policy based on Bayesian networks will be referred to as the *network policy* in the following graphs.

Below, we provide the percentage of farmers victories for each configuration:

Policy	**Victories (%)**
werewolves random policy	47.65
werewolves network policy	38.03

As expected, a more effective werewolves policy leads to a decrease in the chances of victory for the farmers' faction, confirming the superior performance of the network policy compared to the random policy.

Examining the victory percentage plot in relation to the fraction of agents acting as farmers (Figure 10), we can observe a diminishing impact of the werewolves' performance as the relative number of farmers increases. This indicates that the ratio between the populations of farmers and werewolves is the most crucial parameter in this game. In fact, when the population of farmers approaches approximately 80%, the advantages provided by the network policy diminish. This further highlights the positive effect of the farmers' policy depicted in Figure 5, which, in contrast, maintains its advantage over the random policy even with a high percentage of farmers.

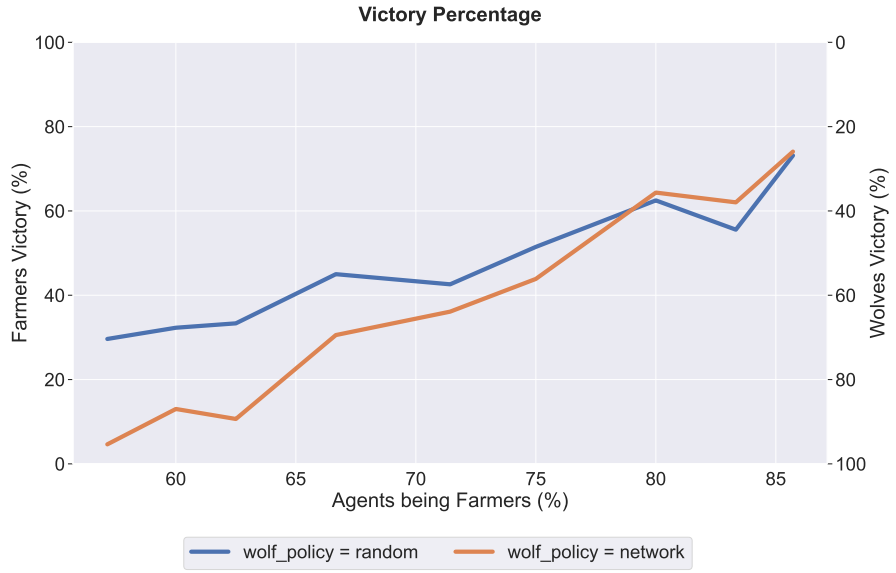

Figure 10: Percentage of victory related to the percentage of total agents in the game being farmers. We compare the average result across all games with werewolves policy set to random and all games with werewolves policy set to Bayesian network policy (network policy).

The correctness plot in Figure 11, which illustrates the accuracy of farmers' votes in relation to the percentage of farmers, provides further validation of the observations made in the previous Section 6.2. Notably, the plot exhibits minimums and maximums occurring at the same intervals, indicating that the correctness of votes is primarily influenced by the proportion of farmers rather than the specific policies employed by farmers or werewolves. The right plot demonstrates an intriguing trend in the werewolves' policy, which, on average, performs worse than random after the second day of the game. This suggests that while the werewolf policy may be more effective in detecting "dangerous" farmers (those who are more likely to have detected werewolves, such as *seers*), their lack of cohesive game strategy and uniform voting patterns may make them easier to detect by the farmers.

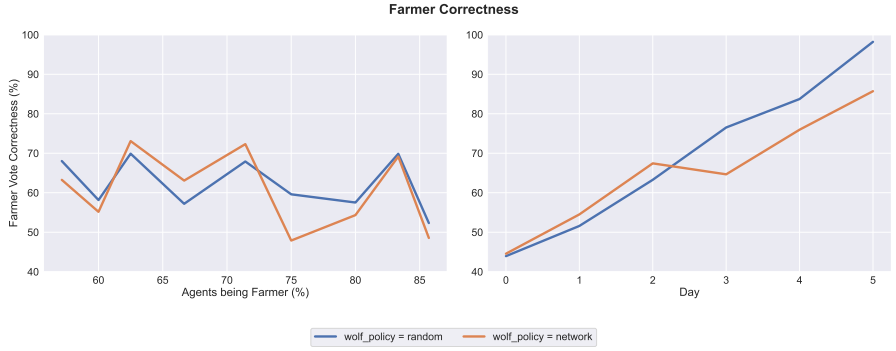

Figure 11: Percentage of farmers voting werewolves, on the left computed w.r.t. the percentage of farmers in the game, on the right w.r.t. the day of the game. We compare the average result across all games with werewolves' policy set to random and all games with werewolves' policy set to Bayesian network policy (network policy).

Upon examining the compactness of the plots in Figure 12, particularly the left plot, we can observe that the compactness varies across different matches when the random policy is employed. However, when the network policy is utilized, the plot demonstrates a more consistent trend, although the effects differ only in the last days of the matches (consider also that these late days are more susceptible to noise due to the limited number of games that reach such durations).

To determine whether the werewolves' policy has a positive or negative impact on the compactness of farmers' votes, further investigation is required. It is necessary to explore the reasons behind the observed trend in the random werewolves policy, which is similarly reflected in Figure 7 for the farmers' policy.

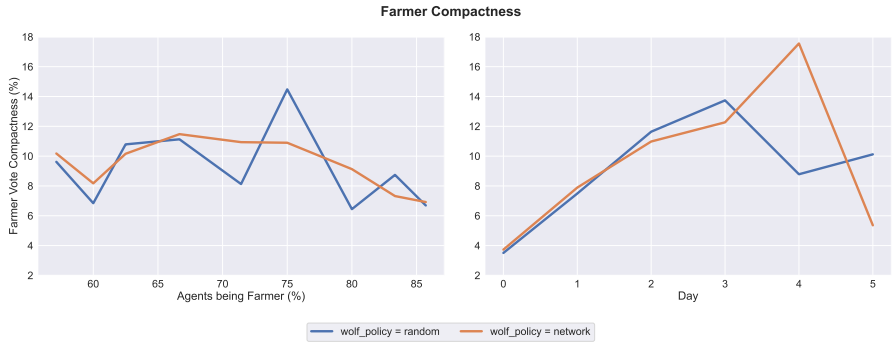

Figure 12: Percentage of farmers voting the same target, on the left computed w.r.t. the percentage of farmers in the game, on the right w.r.t. the day of the game. We compare the average result across all games with werewolves' policy set to random and all games with werewolves' policy set to Bayesian network policy (network policy).

The plots depicted in Figure 13 and Figure 14 provide confirmation that the werewolves' network policy is effective in detecting seers and eliminating them from the game much earlier compared to the random policy.

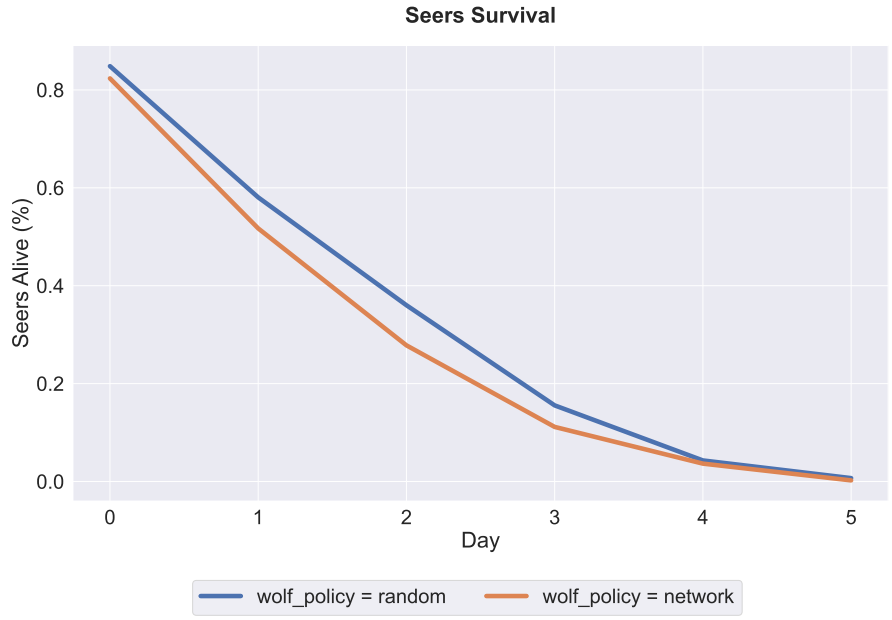

Figure 13: Average percentage of *seers* still alive at certain day of the game. We compare the average result across all games with werewolves' policy set to random and all games with werewolves' policy set to Bayesian network policy (network policy).

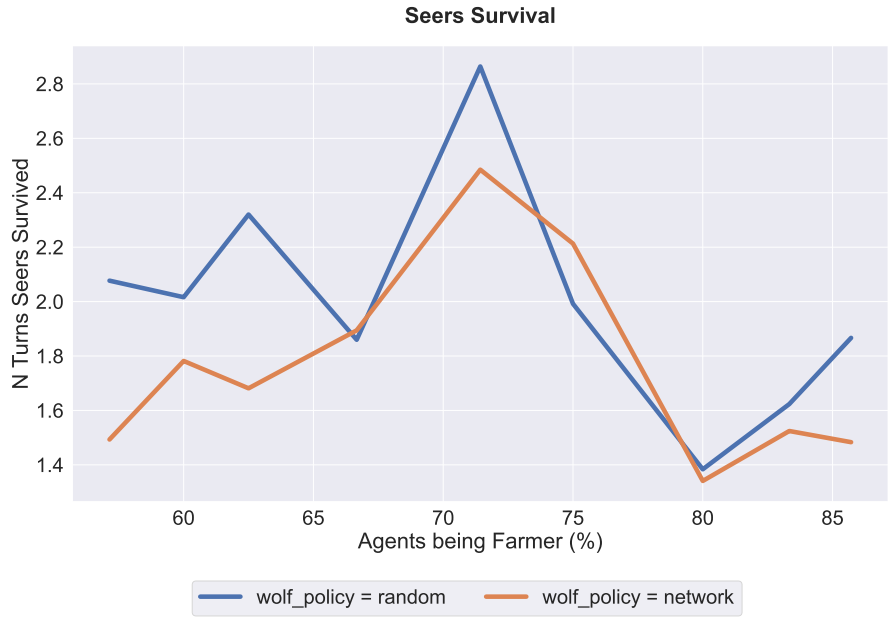

Figure 14: Average number of turns *seers* survive, given the percentage of agents being farmers. We compare the average result across all games with werewolves' policy set to random and all games with werewolves' policy set to Bayesian network policy (network policy).

6.4 UseCase 3 - Perks

As previously mentioned, the perks included in our version of the game are the *seer* and the *bodyguard*. Due to time constraints, we had to limit the number of variable combinations to test. Consequently, we set the maximum number of *seers* and *bodyguards* to 2.

We conducted the same tests as described earlier, and in the following subsections, we will present the results by averaging the measurements across all games with the same number of *seers* or the same number of *bodyguards*. This analysis aims to examine how these perks impact the evolution of the game.

6.4.1 Seers

In this subsection, we categorize the game tests into three groups: those starting with 0 *seers*, those starting with 1 *seer*, and those starting with 2 *seers*.

One notable observation is that the number of *seers* significantly enhances the chances of victory for the farmers. This is evident in the subsequent average percentage of farmers' victories for each configuration:

# *Seers*	**Victories (%)**
0	37.50
1	42.84
2	48.92

The average discrepancy between having 0 or 2 *seers* is about 10%, but from the plot below (Figure 15) we can notice how this difference can be much higher in different situations.

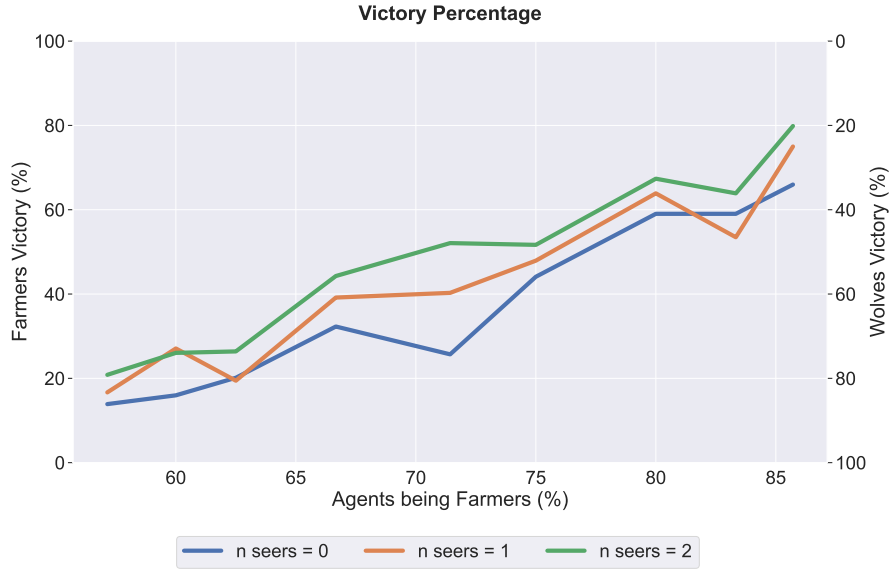

Figure 15: Percentage of victory related to the percentage of total agents in the game being farmers. We compare the average result across all games starting with 0 *seers*, all games starting with 1 *Seers* and all games starting with 2 *seers*.

The correctness, depicted in relation to the percentage of farmers in Figure 16, follows a similar trend as before (Figure 6, Figure 11). However, configurations with 2 *seers* consistently outperform or achieve equal performance compared to configurations with 0 *seers*.

On the other hand, the contribution of *seers* to the accuracy of farmers' votes is more pronounced in the right plot. Configurations with 2 *seers* consistently outperform those with 0 *seers*, with convergence occurring only in the final days of the matches. This can likely be attributed to the fact that the majority of *seers* do not survive beyond 3 days, as indicated by the plot of survived *seers* (Figure 17).

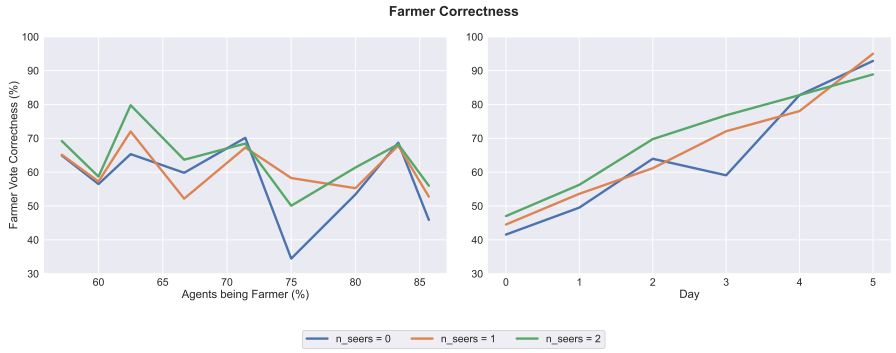

Figure 16: Percentage of farmers voting werewolves, on the left computed w.r.t. the percentage of farmers in the game, on the right w.r.t. the day of the game. We compare the average result across all games starting with 0 *seers*, all games starting with 1 *seers* and all games starting with 2 *seers*.

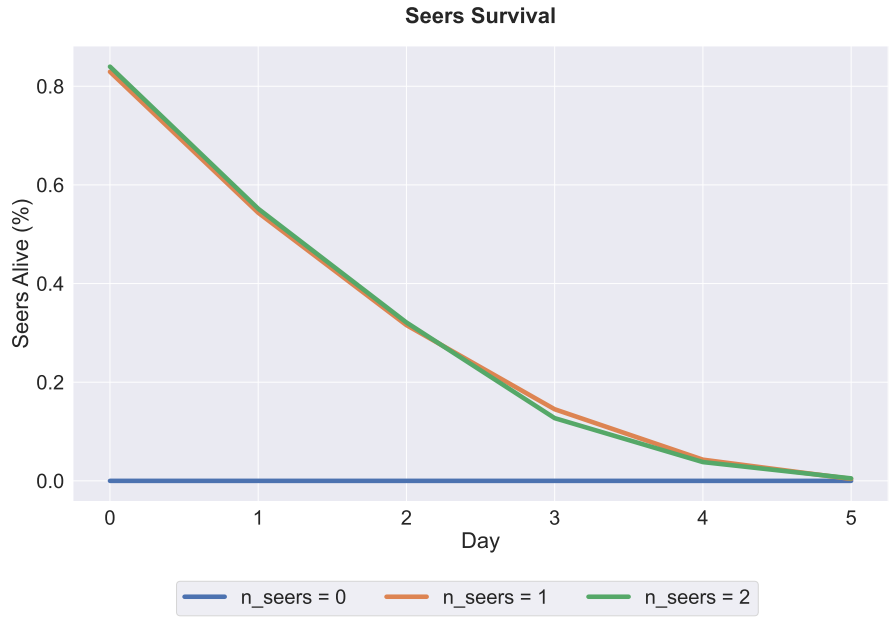

Figure 17: Average percentage of *seers* still alive at certain day of the game. We compare the average result across all games starting with 0 *seers*, all games starting with 1 *seers* and all games starting with 2 *seers*.

Based on the graph below (Figure 18), it appears that configurations with a greater number of *seers* result in lower compactness of farmers' votes. Since the difference is minimal ($\sim$ 4%), it could be attributed to randomness. An alternative explanation could be that *seers*, being the only agents with certain and unequivocal information, make distinct decisions compared to other agents, thereby influencing the level of compactness in the votes. Nonetheless, this behavior is quite unexpected and deserves further

investigation.

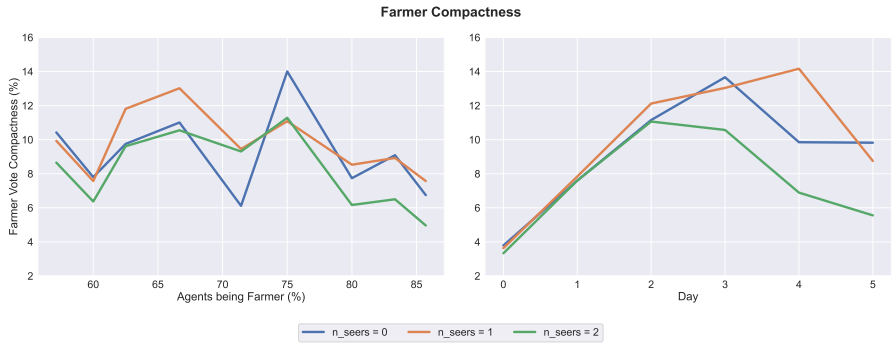

Figure 18: Percentage of farmers voting the same target, on the left computed w.r.t. the percentage of farmers in the game, on the right w.r.t. the day of the game. We compare the average result across all games starting with 0 *seers*, all games starting with 1 *seers* and all games starting with 2 *seers*.

6.4.2 Bodyguards

We conducted an analogous analysis of subsection 6.4.1, but working on the number of *bodyguards* We categorize the game tests into three groups: those starting with 0 *bodyguards*, those starting with 1 *bodyguard*, and those starting with 2 *bodyguards*.

The impact on victory percentage is always positive, but less effective:

# ***Bodyguards***	**Victories (%)**
0	40.03
1	44.34
2	44.40

In contrast to the previous situation, the majority of the impact is evident in the percentages presented above, while the plot below (Figure 19) does not reveal any notable differences among the configurations.

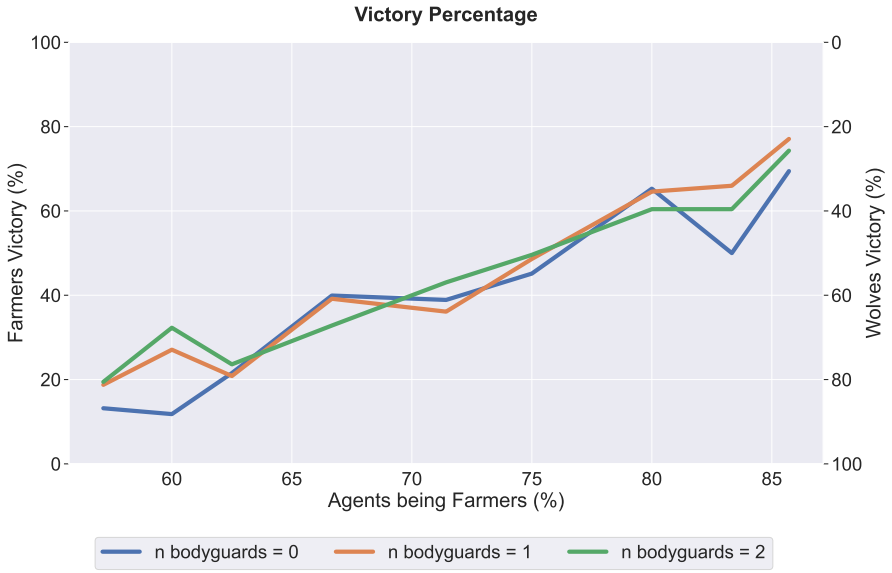

Figure 19: Percentage of victory related to the percentage of total agents in the game being farmers. We compare the average result across all games starting with 0 *bodyguards*, all games starting with 1 *bodyguards* and all games starting with 2 *bodyguards*.

The correctness plot (Figure 20) reveals unexpected behavior, as the *bodyguard* artifact was not expected to have any influence on the agents' beliefs. However, the plot on the right clearly demonstrates a significant negative correlation between farmers' votes (which are based on agents' beliefs) and the number of *bodyguards*. This effect can likely be attributed to the fact that, on average, a game consists less than four farmers. Consequently, a lower percentage of configurations with two *bodyguards* also have two *seers*, which have a much more substantial impact on the game. In specific with 0 *bodyguards* we have on average 1.00 *seers*, with 1 *bodyguards* 0.48 *seers* and with 2 *bodyguards* 0.27 *seers*. In conclusion, the observed decrease in voting accuracy associated with a higher number of *bodyguards* may be due to the inverse relationship between the number of *bodyguards* and the number of *seers* within a given match.

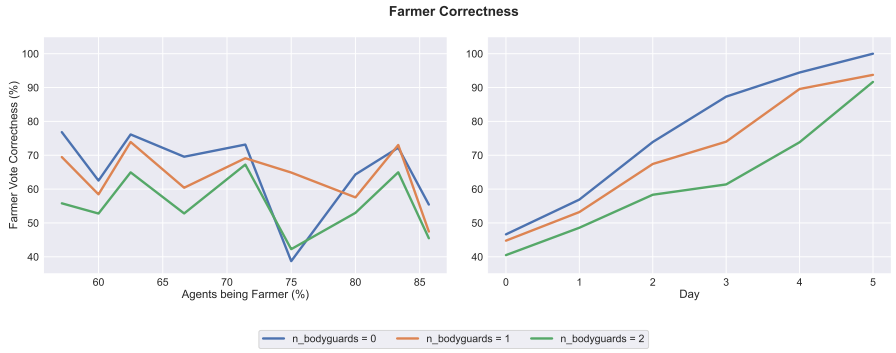

Figure 20: Percentage of farmers voting werewolves, on the left computed w.r.t. the percentage of farmers in the game, on the right w.r.t. the day of the game. We compare the average result across all games starting with 0 *bodyguards*, all games starting with 1 *bodyguard* and all games starting with 2 *bodyguards*.

To further support the previous hypothesis, in Figure 21 we can see how the compactness is much lower when the number of *bodyguards* is reduced, just like it is in Figure 18 when the number of *seers* is higher.

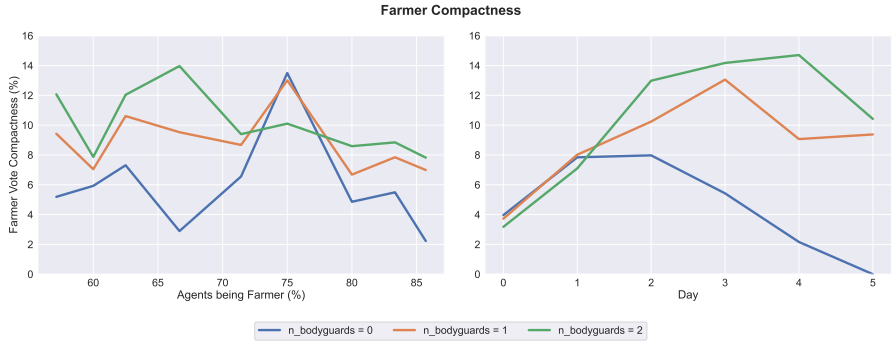

Figure 21: Percentage of farmers voting the same target, on the left computed w.r.t. the percentage of farmers in the game, on the right w.r.t. the day of the game. We compare the average result across all games starting with 0 *bodyguards*, all games starting with 1 *bodyguard* and all games starting with 2 *bodyguards*.

The *bodyguard* appears to fulfill its designated role of protecting the *seer*. This is evident in both Figure 22 and Figure 23, which demonstrates that as the number of *bodyguards* increases, the percentage of *seers* remaining alive each turn also increases.

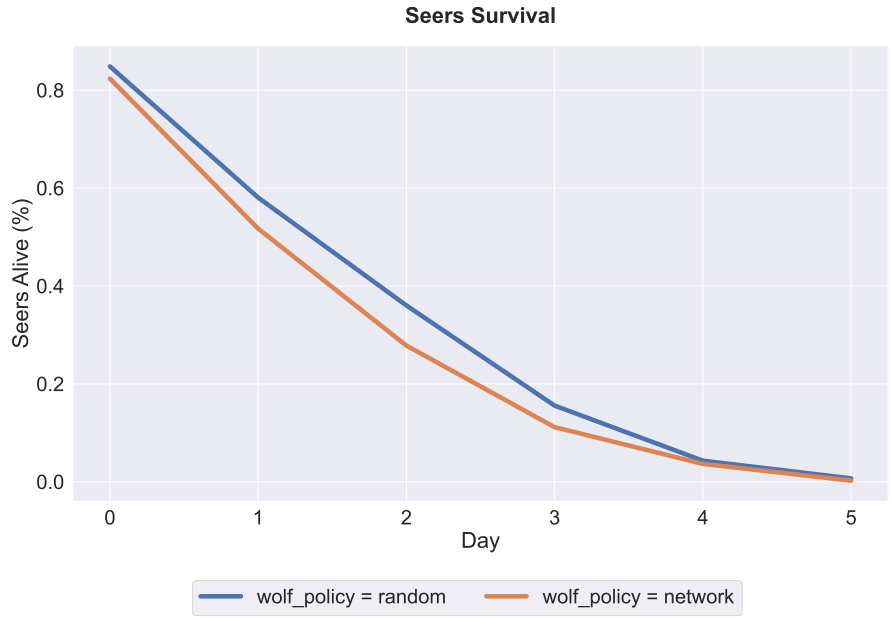

Figure 22: Average percentage of *seers* still alive at certain day of the game. We compare the average result across all games starting with 0 *bodyguards*, all games starting with 1 *bodyguard* and all games starting with 2 *bodyguards*.

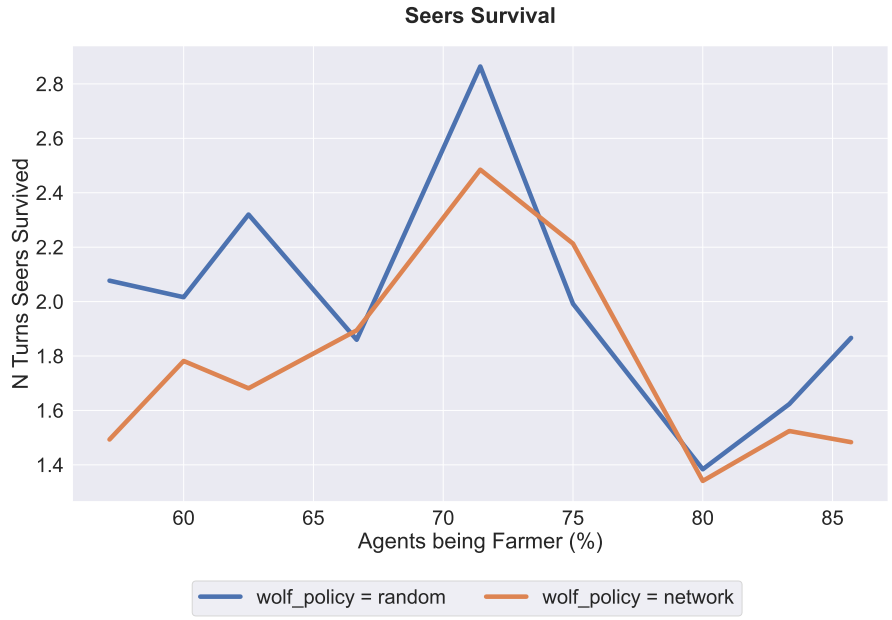

Figure 23: Average number of turns *seers* survive, given the percentage of agents being farmers. We compare the average result across all games starting with 0 *bodyguards*, all games starting with 1 *bodyguard* and all games starting with 2 *bodyguards*.

6.4.3 No perks

Lastly, we compare configurations without perks to the average of all other configurations, which include at least one *seer* or one *bodyguard*.

Configurations without any perks naturally have lower win rates:

# Perks	Victories (%)
0	35.80
> 1	43.78

The difference is inhibited by the configurations with only *bodyguards*, that do not have a significant impact on the victory percentage, as demonstrated in Section 6.4.2. This is further corroborated by the subsequent plot, which shows some data points where the difference is minimal.

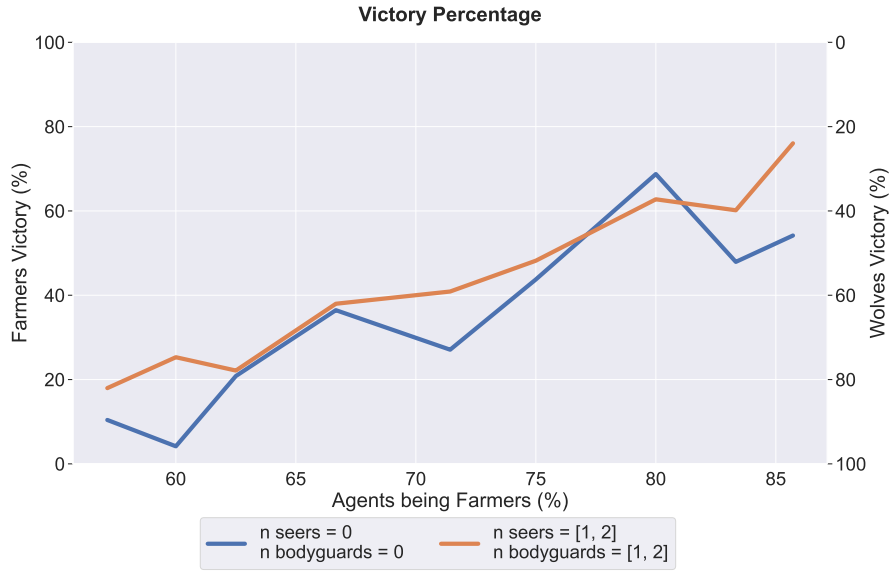

Figure 24: Percentage of victory related to the percentage of total agents in the game being farmers. We compare the average result across all games starting with no perks and all games starting with at least one *seer* or one *bodyguard*.

The correctness depicted in Figure 25 aligns with expectations based on the left plot, where configurations without perks consistently perform worse than the others. In contrast, the right plot reveals a surprising trend: on average, on day 2 and 3, farmers' accuracy is unexpectedly higher when they vote randomly.

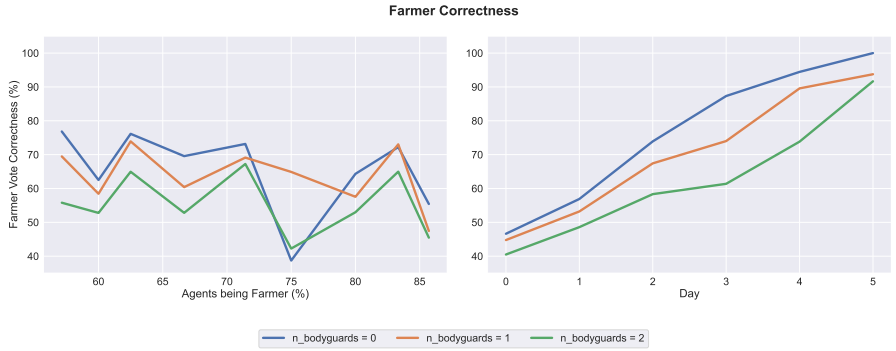

Figure 25: Percentage of farmers voting werewolves, on the left computed w.r.t. the percentage of farmers in the game, on the right w.r.t. the day of the game. We compare the average result across all games starting with no perks and all games starting with at least one *seer* or one *bodyguard*.

The compactness in Figure 26 is much more in line with the expectations, as configurations with at least one perk consistently exhibit better compactness.

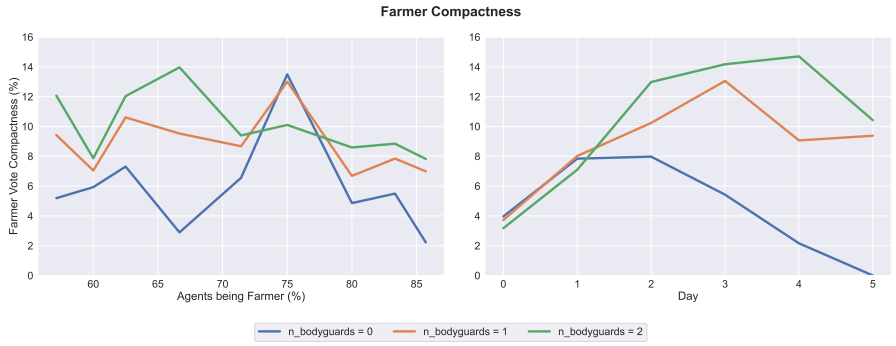

Figure 26: Percentage of farmers voting the same target, on the left computed w.r.t. the percentage of farmers in the game, on the right w.r.t. the day of the game. We compare the average result across all games starting with no perks and all games starting with at least one *seer* or one *bodyguard*.

It is obviously meaningless to compare the average lifespan of *seers* with this configuration sets.

7 Conclusion

The results achieved in this multi-agent system project are largely aligned with our initial expectations.

Specifically, the implementation of Bayesian network policies effectively enhanced the victory rates of the corresponding factions. The introduction of the *seer* artifact resulted in improved accuracy in farmers' votes, thereby positively impacting both the victory rates and the overall performance. Similarly, the *bodyguard* artifact successfully extended the lifespan of the *seers*.

The metric that posed the greatest challenge was compactness, and we also faced difficulties in fully understanding the correctness metric during our analysis across the number of *seers* and *bodyguards*. Once we identified the impact of each variable we tested on the results, we realized the importance of examining the proportions of values for a particular parameter to further investigate any suspicious or unexpected outcomes. For instance, the peculiar findings from the experiments on the number of *bodyguards* in Section 6.4.2 were attributed to variations in the proportion of *seers* in the matches used to calculate the average results. Similar analyses can be conducted to unravel the underlying causes behind all the unexpected behaviors encountered in the experiments outlined in Chapter 6.

7.1 Future Works

As previously mentioned, there are several potential improvements that can be made regarding the metrics and results. To investigate unusual behaviors further, it would be beneficial to examine the populations of matches used for averaging and expand the range of metrics employed to provide a more comprehensive analysis.

Turning to the results themselves, the poor performance of the farmers' policy with pre-training could be significantly enhanced by implementing a Bayesian network that allows for incremental training. Additionally, it would be interesting to observe how the behavior of agents changes with the introduction of other perks such as the *Gunner* or *Assassin*, or any other perks from the real game.

In terms of game implementation, there are a couple of ready-to-be-implemented improvements that are currently not in use for debugging purposes. Firstly, message encryption is a crucial aspect of code security. For instance, given that werewolves communicate during the night, and these messages should not be readable by farmers, we have already created separate channels to ensure privacy. However, employing a different encryption method for messages would further enhance security practices. Additionally, the UUIDs (unique identifiers) of agents should also be changed. Currently, they are clearly correlated with their respective factions, which poses a security risk as farmers have access to the list of agents' UUIDs and could detect directly from there the werewolves. Although they do not have access to the mapping of UUIDs to roles, encrypting the UUIDs would be a better implementation practice to ensure overall security.

7.2 What did we learn

Through this project, we encountered several challenges inherent to multi-agent systems in the form of a dynamic game with varying numbers of agents. We gained valuable experience in handling agents' *beliefs* in different scenarios, as farmers possessed individual knowledge while the werewolves shared the same information among their faction members.

The game structure presented us with the task of addressing communication issues, compounded by the need to conceal werewolves' nocturnal exchanges from the farmers. To tackle this, we devised an encryption mechanism and we learnt how to divide communications into distinct topics.

Additionally, we honed our skills in implementing various artifacts, ranging from the straightforward *bodyguard* to the more intricate *seer*, whose effects extended to the beliefs of agents interacting with the artifact.

References

[1] osBrain. osbrain — general-purpose multi-agent system module. https://osbrain.readthedocs.io, 2023. [Online; accessed 3-June-2023].

[2] pgmpy. pgmpy — implementation for bayesian networks with a focus on modularity and extensibility. https://pgmpy.org/, 2023. [Online; accessed 3-June-2023].

[3] Pyro. Pyro — library that enables you to build applications in which objects can talk to each other over the network, with minimal programming effort. https://pyro4.readthedocs.io, 2023. [Online; accessed 3-June-2023].

[4] Wikipedia. Expectation–maximization algorithm — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Expectation%E2%80%93maximization%20algorithm&oldid=1152010048, 2023. [Online; accessed 04-June-2023].

[5] Wikipedia. Bayesian network — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Bayesian%20network&oldid=1157714124, 2023. [Online; accessed 04-June-2023].

[6] Wikipedia. Encapsulation (computer programming) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Encapsulation%20(computer%20programming)&oldid=1156567264, 2023. [Online; accessed 04-June-2023].

[7] Wikipedia. Conditional probability distribution — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Conditional%20probability%20distribution&oldid=1142648522, 2023. [Online; accessed 04-June-2023].

[8] Wikipedia. Multi-agent system — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Multi-agent%20system&oldid=1153395112, 2023. [Online; accessed 04-June-2023].

[9] Wikipedia. Publish–subscribe pattern — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Publish%E2%80%93subscribe%20pattern&oldid=1157505337, 2023. [Online; accessed 04-June-2023].

[10] Wikipedia. Universally unique identifier — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Universally%20unique%20identifier&oldid=1158411382, 2023. [Online; accessed 04-June-2023].

[11] Wikipedia. Variable elimination — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Variable%20elimination&oldid=1064123265, 2023. [Online; accessed 04-June-2023].