



SAPIENZA  
UNIVERSITÀ DI ROMA

## Reinforcement Lupus

Cooperation through emergent communication in *The Werewolf* social deduction game

Dipartimento di Ingegneria informatica, automatica e gestionale

Corso di Laurea Magistrale in Artificial Intelligence and Robotics

### Candidate

Nicolo' Brandizzi

ID number 1643869

### Thesis Advisors

Prof. Luca Iocchi

Prof. Davide Grossi

### Co-Advisor

Prof. Roberto Capobianco

Academic Year 2019/2020

---

**Reinforcement Lupus**

Master's thesis. Sapienza – University of Rome

© 2020 Nicolo' Brandizzi. All rights reserved

This thesis has been typeset by L<sup>A</sup>T<sub>E</sub>X and the Sapthesis class.

Author's email: [brandizzi.1643869@studenti.uniroma.it](mailto:brandizzi.1643869@studenti.uniroma.it)

*Dedicated to my father.  
Grazie di tutto.*



## Abstract

Multi agent deep reinforcement learning (MADRL) has been studied intensively for its ability to develop complex behaviours from a simple set of rules. One of its field is the analysis of emergent communication in articulated environment such as social deduction games.

In this thesis we investigate if and how various form of communication channels influence the outcomes of the Werewolf deduction game with deep RL <sup>1</sup>. We first study the game with a wide range of mathematical models drawing the basis for the later analysis.

Then we show how the introduction of a communication signal greatly increases the winning rate of the players for two different kind of settings; in particular we analyze the results for diverse game logic and prove that each one is positively influenced by said signal.

Finally we study the effect of the channel's length and range on the overall performance and prove a non linear relationship between the two.

---

<sup>1</sup>The code is available at [https://github.com/nicofirst1/rl\\_werewolf](https://github.com/nicofirst1/rl_werewolf)



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	1
1.2	Related work . . . . .	2
1.3	Proposed Work . . . . .	3
1.4	Structure . . . . .	3
<b>2</b>	<b>Methods</b>	<b>5</b>
2.1	The Game . . . . .	5
2.1.1	Rules . . . . .	5
2.1.2	Game theory . . . . .	6
2.2	Theoretical framework . . . . .	7
2.2.1	Fully Connected Network and Embeddings . . . . .	7
2.2.2	Memory Persistence . . . . .	9
2.2.3	Markov decision process . . . . .	11
2.2.4	Reinforcement Learning . . . . .	12
2.3	Environment . . . . .	15
2.3.1	Tool-kits and Libraries . . . . .	15
2.3.2	Spaces . . . . .	16
2.3.3	Rewards . . . . .	18
2.3.4	Workflow . . . . .	19
2.3.5	Metrics . . . . .	22
2.3.6	Parametric Action Wrapper . . . . .	22
2.3.7	Evaluation Wrapper . . . . .	23
2.3.8	Example run . . . . .	23
2.4	Policies . . . . .	25
2.4.1	Model policies . . . . .	25
2.4.2	Static . . . . .	25
2.4.3	Trainable . . . . .	26
<b>3</b>	<b>Results</b>	<b>29</b>
3.1	Baselines . . . . .	29
3.1.1	Theoretical . . . . .	30
3.1.2	Practical . . . . .	31
3.1.3	Expanded Tree . . . . .	33
3.2	Nine Players . . . . .	33
3.2.1	No communication . . . . .	34

3.2.2	Bit communication . . . . .	39
3.2.3	Multi channel communication . . . . .	43
3.3	Twenty one players . . . . .	47
3.3.1	No communication . . . . .	48
3.3.2	Bit communication . . . . .	50
3.3.3	Multi channel communication . . . . .	51
<b>4</b>	<b>Discussion</b>	<b>59</b>
4.1	Nine Players . . . . .	59
4.2	Twenty-one Players . . . . .	60
	<b>Bibliography</b>	<b>63</b>

# Chapter 1

## Introduction

### 1.1 Background

Multi agent system (MAS) have been intensively studied for a wide variety of domains such as collective robotics, game theory, distributed control and telecommunication. In her book, [Sycara, 1998] gives an extended analysis on the application and architectures of such systems, including a state-of-the-art theoretical background. The scope of MAS is to build a network of intelligent agents interacting with each other to achieve a goal which can be common or not; this requires the agents to learn from past experiences in a fast, robust fashion effectively excluding the usage of hand-coded fixed procedures.

This kind of behaviour is what reinforcement learning (RL) aims to achieve. RL was born by the union of two fields of study: learning from trials and error in the animal domain in the mid 80s and optimal control through value functions and dynamic programming in the 50s. Although there is no precise point in which modern RL came to be, the work of Harry Klopff, [Klopff, 1972], is considered the one that associated RL with artificial intelligence introducing the world to a new area of research.

In this context, reinforcement learning was used mainly for playing simple games, such as checkers [Samuel, 1959], or in trivial robotics control problems as described in [Kaelbling et al., 1996].

Given the hunt for adaptable agents in multi agent systems, RL seemed to be the perfect candidate for the task. In their work [Buşoniu et al., 2010], the authors outline the scope of a multi agent reinforcement learning (MARL) system, and how it can be used to model a wide variety of social systems found in nature and in modern society <sup>1</sup>. In these settings the agents behavior can be cooperative, competitive or a mix of the two; in [Buşoniu et al., 2010] a study of these different kinds of settings and the algorithms associated with them is pursued while [Tan, 1993] focuses on the difference between competitive and cooperative agents.

---

<sup>1</sup>From food-collective ants systems to complex sharing of information in social networks.

Although MARL has been used for a wide variety of application, the reinforcement learning paradigm was not strong enough to be applied to complex social system. For this reason, researches such as [Schmidhuber, 2015], tried to merge together deep neural networks with RL creating Deep-RL (DRL). This new technique allows RL to scale to problems that were previously intractable such as playing video games using the pixels as input [Mnih et al., 2013]. Shortly after, this practice was introduced to the multi agent systems to study the complex emergent behavior of multiple agents interacting with each other to reach a goal.

## 1.2 Related work

This research areas are strictly tied to the study of complex behavior arising from the interaction of simple agents; these aspects are mainly studied through the usage of complex game systems.

In their work [Baker et al., 2019] show how a few simple rules from the *Hide'n Seek* game can generate complicate behaviors to the point of exploiting environmental errors to their advantages. On the same line, [Leibo et al., 2019] carries out an extensive analysis on the problem of autocurricula and non stationary learning in multi-agent deep reinforcement learning (MADRL); they point out how the interaction of competitive agents can culminate in an endless cycle of counter strategies.

One aspect of complex behavior in a multi-agent system is the emergence of a communication language both for cooperation and coordination. The workshop of Emergent Communication ([emecone](#)) sees many publication in the field of natural language processing (NLP) strictly tied to MADRL and social deduction games as environment. One such game is [Werewolf](#) where the players find themselves split into two opposite groups in a partially known environment. This game has gained increased popularity, in the field of cooperation through emergent communication, especially in Japan where the annual [AiWolf](#) context sees artificial agents competing with and against human players to win the game with a fixed language syntax.

In these settings the problem of who to trust is the main point which requires intensive research. In [Raileanu et al., 2018], the agents' own policy as a predictive model of the other agents' behavior. On the same wavelength, [Nakamura et al., 2016] use a similar approach in the Werewolf game; the authors combine two neural network which aim is to model the other agents viewpoint from their and the other prospective.

Finally [Wang and Kaneko, 2018] use a Deep Q-Network to choose who to trust or to kill in the AiWolf environment.

## 1.3 Proposed Work

All the aforementioned studies define a fixed communication syntax and let the DRL agents work their way to optimize such language. We present a new framework for the Werewolf game in which the channel regulating the information exchange can vary depending on the configuration.

Our aim is to study if and how various communication systems can influence the outcome of the game in which only one party is able to learn while the other one has a fixed, hand-coded policy.

Our hypothesis is that, under the same circumstances, the agents which are able to communicate will perform much better than the ones not allowed to exchange information. Moreover, we speculate that different kind of communication settings will have diverse influences on the amount of coordination between the agents as well as the final outcome of the game.

## 1.4 Structure

In [Chapter 2](#) we first give a brief description of the Werewolf social deduction game, [2.1](#), in which the rules, roles and phases are explained; moreover a brief statistical analysis of the game is carried out in [2.1.2](#).

Next we move to the theoretical background ([Section 2.2](#)) of the components of the system we propose: neural networks ([2.2.1](#)), LSTMs ([2.2.2](#)), Markov decision processes ([2.2.3](#)) and its direct descendant, reinforcement learning ([2.2.4](#)).

Moving on, the environment structure is introduced ([Section 2.3](#)) in which we first reference the RL basic building blocks, e.g. the spaces ([2.3.2](#)) and the reward function ([2.3.3](#)), and then explain in great detail the environment's workflow ([2.3.4](#)) and the metrics used to evaluate the learning ([2.3.5](#)).

Later a brief description of the model ([2.4.1](#)) is given, followed by a more accurate characterization of the different agents' policies ([Section 2.4](#)); in particular the different kind of static policies ([2.4.2](#)) and the only trainable one ([2.4.3](#)).

In [Chapter 3](#) we present the results of the thesis.

We start with a baseline section ([Section 3.1](#)) in which three different baselines are introduced; first a theoretical approach ([3.1.1](#)) based on [2.1.2](#) is given, then the baselines' values are extracted directly through the usage of the implemented policies in [3.1.2](#) and finally the expanded tree of possibilities accurately depict each branch probability in [3.1.3](#).

We then present results for a nine players game are given ([Section 3.2](#)). In this game three kind of communication channels are compared, namely: no communication ([3.2.1](#)), bit communication ([3.2.2](#)) and multi channel communication ([3.2.3](#)).

Finally twenty one players game is analyzed in [Section 3.3](#); like in the previous

section, we follow a similar structure where first the no communication setting is treated (3.3.1), then the bit one (3.3.2) and finally the multi channel (3.3.3).

In conclusion, in [Chapter 4](#), we discuss the results achieved in the previous section. We first focus on the nine player setting ([Section 4.1](#)) where every combination of policies is treated; then shift to the twenty one players environment ([Section 4.2](#)) and proceed in a similar fashion.

# Chapter 2

## Methods

### 2.1 The Game

Werewolf<sup>1</sup> is a social deduction game modeling conflicts between two groups in a partially known environment. The game was born in the spring of 1987 at the psychology department of the Moscow State University and spread through the world acquiring many different names; some of them are listed below:

- Mafia (Mafia, International)
- Lupus in fabula (Wolf from fable, Latin)
- Pueblo duerme (Sleeping villagers, Spain)
- Los Hombres Lobo de Castronegro (The Werewolves of Castronegro, Spain)
- Μια Νύχτα στο Παλέρμο (One night in Palermo, Greek)
- Městečko palermo (Town of Palermo, Check)
- 狼人杀 (Werewolf kill, Chinese)
- Libahunt (Werewolf, Estonia)
- Loup Garous (Werewolves, French)
- Werewölfe (Werewolves, German)
- Weerwolven (Werewolves, Dutch)

#### 2.1.1 Rules

In its easiest version, the game sees two groups, villagers and werewolves. The wolves know exactly the identity of each player, while the villager are certain exclusively about their role and the number of werewolves. In an open setup an additional moderator is needed to coordinate the players.

The game is divided into two phases: night and day, interleaving each other.

---

<sup>1</sup>The original name for the game is [Mafia](#).

**Night** At the start of the game, every player closes their eyes and the moderator allows the wolves to see each other and to vote for a victim among the sleeping villagers. This victim will be the one eaten by the wolves.

**Day** At day time, everyone opens their eyes and the moderator announces the recently killed villager which now has to stay silent for the rest of the game. At this point each player is given some time for his/her harangue and then everyone vote for a player to execute.

The identity of the executed player is not revealed until the end of the game.

**Game end** The game ends either when the villagers execute the last werewolf or there is an even number of both roles. The latter case implies the wolves winning since the execution phase can be stalled, thus taking away the only possibility for villagers to kill the wolves.

### 2.1.2 Game theory

The Werewolf game has been modeled in an optimal setting in which both the villagers and the werewolves use a randomized strategy. In this context the werewolves  $w$  are the square root of the total number of players  $n$ ; [Braverman et al., 2008] showed that the werewolves chance to win in such a setting is approximately:

$$W(w, n) \approx \frac{w}{\sqrt{n}}$$

Moreover, in [Migdal, 2010], the authors point out how the parity of the number of player strongly influences the game results. In particular they showed that adding one villager to an even number of players increases the werewolf-winning chances more <sup>2</sup> than dropping a villager. This counter intuitive demonstration culminates with a closed form formula for the werewolf winning probability:

$$W(w, n) = 1 - \sum_{i=0}^w \binom{w}{i} (-1)^i \frac{(n-i)!!}{n!!(n \bmod 2 - 1)!!} \quad (2.1)$$

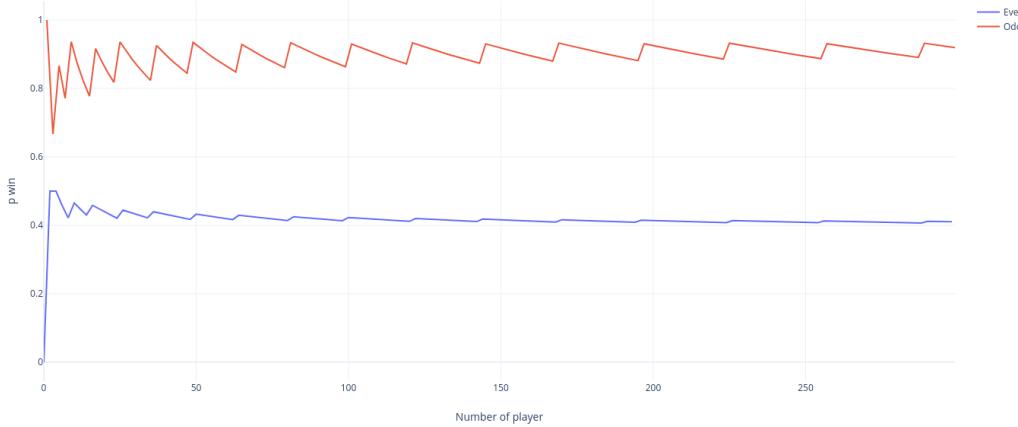
To better understand the importance that parity has on the winning chances of werewolves, Figure 2.1 shows how steep this difference can become.

It is interesting to notice how both the odd and even probabilities seem to have an oscillating behaviour for peaks that wanes proportionally to the number of players. The odd numbers seems to be more influenced by this propriety, varying the winning chances by a  $\pm 20\%$  factor. More research should be done in order to accurately study the winning function and to determine the reason behind the oscillating trend.

These results will be later used, together with Equation 2.1, to draw the baseline of the environment in Section 3.1.

---

<sup>2</sup>The rise factor is approx.  $\sqrt{\pi/2}$



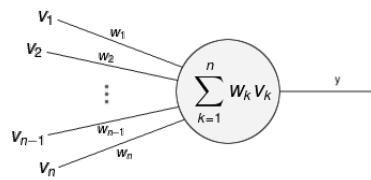
**Figure 2.1.** Winning probability for werewolves as given by [Equation 2.1](#)

## 2.2 Theoretical framework

In the following section the basic concept of reinforcement learning (RL) and neural network (NN) will be briefly explained to get the reader familiar with the terminology.

### 2.2.1 Fully Connected Network and Embeddings

To better understand the model structure in [Subsection 2.4.1](#) and the concept of Deep RL, a brief introduction on the mathematics of a fully connected network (FCN) and an embedding must be presented.



**Figure 2.2.** Artificial neuron

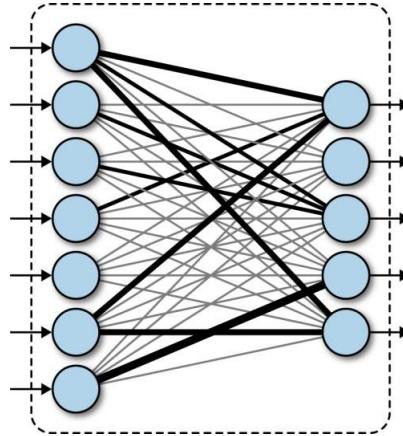
**Neuron** A FCN is made up by layers which are made up of neurons, thus the definition of neuron should come first in the explanation.

The notion on neuron is first found in [\[McCulloch and Pitts, 1943\]](#), where the authors give an early mathematical modeling of the human brain. They argued that

neurons were capable of computing arbitrary function on boolean inputs.

As for [Figure 2.2](#), given  $n$  inputs  $v_0, v_1, \dots, v_n$  and weights  $w_0, w_1, \dots, w_n$ , the neuron's output  $y$  is a weighted sum.

**Fully connected layer** A fully connected layer (FCL), [Figure 2.3](#), with  $n$  inputs and  $m$  outputs can be seen as a function  $\mathcal{R}^n \rightarrow \mathcal{R}^m$ .



**Figure 2.3.** Fully connected layer <sup>3</sup>

By defining  $X \in \mathcal{R}^n$ , the layer's input, and  $X \in \mathcal{R}^m$ , the output, the  $i - th$  element of  $Y$  is given by:

$$y_i = \sigma(w_{1i}x_1 + w_{2i}x_2 + \dots + w_{ni}x_n)$$

Where  $w_i$  is the  $i - th$  weight associated with  $x_i$  and  $\sigma$  is a non linear function, called activation function (typically  $\tanh$  <sup>4</sup>). It is trivial too see how stacking the outputs together yields:

$$\begin{bmatrix} x_1 & \dots & x_n \end{bmatrix} = X = W \cdot Y = \begin{bmatrix} w_{11} & \dots & w_{1m} \\ \vdots & & \vdots \\ w_{n1} & \dots & w_{nm} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

**Fully connected network** A fully connected network is a simple horizontal stack of FCLs as per [Figure 2.4](#).

The depth of the network is defined by the number of FCLs stacked, indeed the definition of deep learning derives from the horizontal depth a network has.

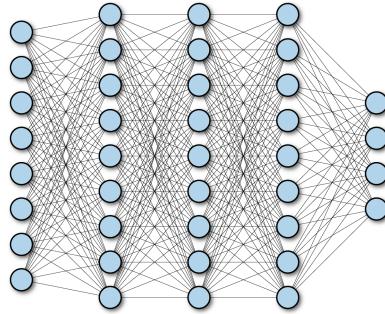
**Embeddings** Formally an embedding is a mapping of a discrete variable in a vector of continues numbers. The use of such structures is of fundamental value for fields like NLP <sup>6</sup>, where the dimensionality of the input is such that modern

<sup>3</sup>Image taken from [\[Tammina, \]](#).

<sup>4</sup>For an extensive comparison between activation functions refer to [\[Karlik and Olgac, 2011\]](#).

<sup>5</sup>Image taken from MIT course: Deep Learning for Computer Vision, Ava Soleimany

<sup>6</sup>Natual Language Processing



**Figure 2.4.** Fully connected network <sup>5</sup>

computers are not able to enumerate all their possible combinations. The easiest type of embeddings is the one hot encoder (OHV) which simply encodes each input in a boolean variable of adequate size; its limitation are trivial since, for large input dimensions, the OHV yields an extreme length.

### 2.2.2 Memory Persistence

It is trivial to understand that the agents need some kind of memory to correctly play the game; in fact, recalling what other players did in the previous phases of the game is crucial to accurately guess a player's role.

**Recurrent Neural Network** The history of Recurrent Neural Network (RNN) and how they work is not in the scope of this project, the interested reader is referred to [Schuster and Paliwal, 1997].

For the time being the only necessary information about RNN is that they are the first attempt to tackle the problem of vanishing memory in neural networks; they work essentially as a standard neural network whose output is then fed back into itself. This internal loop system allows the network to keep a reference of the current and past states so to retain important pieces of information.

The main issues with this kind of architecture are two:

- Persistence of information in long sequences: as mentioned before, for long enough inputs, e.g. entire paragraphs of text, the network is not able to retain knowledge acquired far earlier in the process.
- Vanishing gradient: the shrinking of the gradient as it back-propagates through time. When the gradient reaches a small enough value then its contribution becomes zero and the learning stops.

To address this two problems, a system of gates which regulates the flow of information was created. The two most famous are the gated recurrent unit GRU, [Cho et al., 2014], and the long-short time memory (LSTM), [Hochreiter and Schmidhuber, 1997] architectures <sup>7</sup>; the latter will be studied in the following paragraph.

---

<sup>7</sup>For an extensive analysis of different RNN architectures, see [Chung et al., 2014].

## LSTM

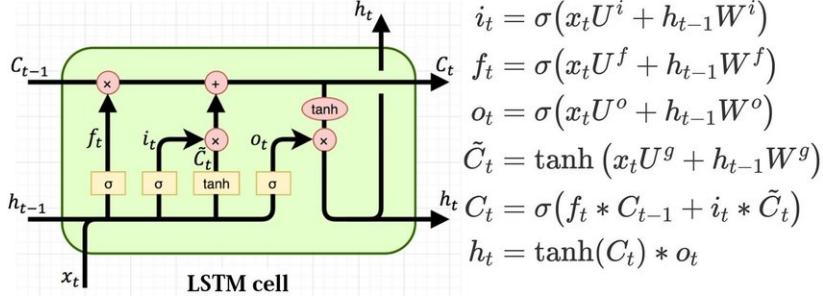


Figure 2.5. LSTM gate architecture <sup>8</sup>

The capacity of LSTMs to learn long-term dependencies in the input is given by their gate structure reported in Figure 2.5. As can be seen from the architecture an LSTM cell has three inputs: the previous cell state  $C_{t-1}$  is passed to the current cell, processed and passed to the next cell. The processing is done in two separate instances reported in the following paragraphs.

**Forget Gate** The first modification of the previous cell state is done through a multiplication with the so called *forget state*. This gate takes as input the cell's input  $x_i$ , e.g. a word, and the previous hidden state which acts as the neural networks memory. These two inputs are multiplied with their relative weight matrices  $W^f, U^f$  and passed through a sigmoid function  $\sigma$ . The output is:

$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$

**Input Gate** The next step is to compute the input gate's values. This gate has the ability to choose which new information to store in the cell state and it is split in two parts: the first part is called the *input gate layer* and it uses the concatenation  $[h_{t-1}, x_t]$  multiplied with another pair of weights  $U^i, W^i$  and passed in the sigmoid function  $\sigma$  to decide which parts of the previous input input to keep  $i_t$ .

The next step is to compute a vector of new candidate values  $\tilde{C}_t$  through the usage of the tanh function and another set of weights  $U^g, W^g$ ; this part conceptually holds the new information that was considered worthy to be kept and passed onto the next cell.

**Output Gate** Finally the cell decides what to output with the *output gate*. This gate computes the usual sigmoid function on the weighted sum:

$$o_t = \sigma(x_t W^o + h_{t-1} W^o)$$

Then  $o_t$  is multiplied with the tanh of the current cell state to form the final hidden cell state:

$$h_t = o_t \cdot \tanh(C_t)$$

Where

$$C_t = \sigma(f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t)$$

<sup>8</sup>Image taken from [medium](#)

### 2.2.3 Markov decision process

To have a comprehensive understanding of the RL's mechanics, we must refer to Markov decision processes (MDPs) and how they shaped the world of game theory.

An MDP is based on the *Markov property* which states:

*"The future is independent of the past given the present"*

This property can be mathematically interpreted as follows:

$$P[\bar{s}_{t+1}|s_t] = P[\bar{s}_{t+1}|s_1, \dots, s_t]$$

Where  $P[\bar{s}_{t+1}|s_t]$  is the probability to transition to a new state  $\bar{s}_{t+1}$  at time  $t + 1$ , given the current state  $s_t$  which is equal to the probability to transition to the next state given the entire history of the states  $s_1, \dots, s_t$ .

In general a MDP is a tuple of four elements  $(S, A, P_a, R_a)$  where:

- $S$  is a finite set of states.
- $A$  is a finite set of actions.
- $P_a(s_t, \bar{s}_{t+1})$  is the probability to transition from state  $s_t$  to a new state  $\bar{s}_{t+1}$  given action  $a$ .
- $R_a(s_t, \bar{s}_{t+1})$  is the reward associated to the previous transition.

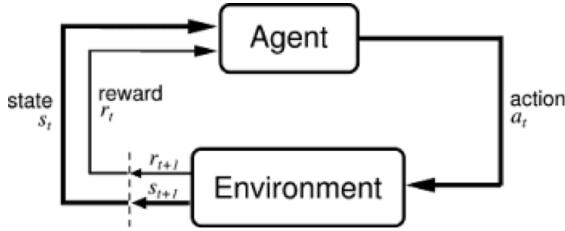
It is now trivial to see how the problem reduce to find some policy  $\pi(s)$  to maximize  $R_a(s_t, \bar{s}_{t+1})$ ; once such policy <sup>9</sup> is found the system behaves as a Markov chain [Gilks, 2005].

**Value learning** Finding the correct policy depends on the field of application. For classical MDPs the approach is to maximize the expected discounted sum over the infinite horizon <sup>10</sup>:

$$V^\pi(s_t) = E \left[ \sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \right]$$

Where the policy decides the action to take  $a_t = \pi(s_t)$ , given the current state  $s_t$ . The discount factor  $0 \leq \gamma^t \leq 1$  determine the weight future rewards have on the current estimation <sup>11</sup>.

Since the summation is performed on an infinite horizon we must enforce the condition  $\gamma < 1$ .



**Figure 2.6.** Basic RL schema

### 2.2.4 Reinforcement Learning

The history of reinforcement learning (RL) sees it as an extension of the MDP. The most basic version of RL is made up of two functions:

- An *environment* that transforms an action taken in  $s_t$  into a tuple (new state, reward):

$$E(s_t, a_t) = (\bar{s}_{t+1}, r_{t+1})$$

- An *agent* that takes a state and a reward and outputs a new action:

$$A(s_t, r_t) = a_t$$

By connecting the two functions the results is a feedback loop as in [Figure 2.6](#). Again the aim of the agents is the same as the one in the MDP, that is maximize the reward.

**Quality Learning** The simplest type of algorithm used in RL is Q-learning [[Watkins and Dayan, 1992](#)].

The latter belongs to the category of off-policy algorithms because the action taken are outside the current policy <sup>12</sup>. In Q-learning the agent uses a *q-table*,  $Q(a_i, s_j)$ , to store the reward  $r_{ij}$  given by action  $i$  in state  $j$ ; these values are later used when the agents find itself in a state  $k$  where it chooses the action that maximizes the reward in that given state <sup>13</sup>:

$$a^* = \text{argmax}_a Q(a, s_k)$$

It is trivial to see that this kind of learning can be applied to problem where the size of the action and state sets is relatively small.

---

<sup>9</sup>A policy is a system of principles to guide decisions achieving a desired outcome.

<sup>10</sup>In the classical problem the horizon is set to be infinite, but in practical approaches the concept of *soft/finite horizon* is used [[Smallwood and Sondik, 1973](#)].

<sup>11</sup>For  $\gamma = 0$  the policy is short sighted, while for  $\gamma = 1$  the policy is heavily influenced by future rewards.

<sup>12</sup>The difference between on-policy and off-policy might be confusing. Refer to [this](#) for more insights.

<sup>13</sup>This behavior is called *exploiting*. The other kind of procedure is called *exploring* and consists in the agent acting randomly.

**Advantage function and KL-divergence** As previously mentioned, the MDP tries to maximize the expected reward in its policy; for complex tasks which requires a large number of samples, this method can introduce a strong variance in the policy gradient, thus requiring smaller trust regions and higher training time.

In 2015 [Schulman et al., 2015b] introduced the advantage function defined as the difference between the q-value and the expected discounted sum:

$$A^\pi(a_t, s_t) = Q^\pi(a_t, s_t) - V^\pi(s_t)$$

The intuitive idea behind this method is that the *absolute reward may not be as important as an action compares with the average action*.

On the other hand, variance in the learning can be introduced when the difference between policies updates,  $\pi_t$  and  $\pi_{t+1}$ , is too high. For this reason the Kullback–Leibler divergence [Van Erven and Harremos, 2014], or KL for short, is employed to constraint such difference to be less than a given value:

$$KL(\pi_t, \pi_{t+1}) \leq \delta$$

**Policy Network** For complex environment, having a table mapping states to action actions might not be feasible. For such problems a policy network  $\pi_\theta$  (PN) is needed.

A PN uses a FCN to encode a huge number of inputs, e.g. a raw image of  $210 \times 160 \times 3$  pixels for the Pong <sup>14</sup> game, and outputs a vector of probabilities proportional to the action expected to maximize the reward.

**Trust Region Policy** In the following we are assuming that the reader is familiar with the concept of supervised learning with gradient based algorithm <sup>15</sup>.

Since the agent is immediately rewarded for an action taken in the previous step, the well-studied classes of supervised methods can be used to train the FCN. In this case the supervision can be computed with the back-propagation method either at time  $t + 1$ , that is after the action is carried on, or later, depending on the environment specification.

The objective function becomes :

$$\begin{aligned} \max_{\theta} \quad & E_t [r_t(\theta) A_t], \quad A_t = A^\pi(a_t, s_t) \\ \text{s.t. :} \quad & \hat{E}_t [KL(\pi_{\theta_t}, \pi_{\theta_{t+1}})] \leq \delta \end{aligned}$$

Where:

$$r_t(\theta) = \frac{\pi_{\theta_{t+1}}(a_t, s_t)}{\pi_{\theta_t}(a_t, s_t)}$$

In the probability ratio between the old policy  $\pi_{\theta_t}$  and the new one  $\pi_{\theta_{t+1}}$  which measures the difference between the two policies.

---

<sup>14</sup>Refer to [Mnih et al., 2015] for more insight on deep RL with Atari games.

<sup>15</sup>Please refer to [Amari, 1993] for more information.

The downside of using such policy gradient based methods is their high computational intensity in estimating the second-order derivative and its inverse. To tackle this problem, researchers implemented new algorithms to constrain [Schulman et al., 2015a] or optimize [Wang et al., 2016] the size of the policy update, i.e. the trust region. While these methods solve the problem of efficiently computing the back-propagation, they introduce additional complexity to the system and requires more resources to be computed.

**Proximal Policy Optimization** The proximal policy optimization algorithm [Schulman et al., 2017], or PPO for short, relaxes the hard constraint in the policy optimization, chaining them in penalties in the objective function. This effectively allow the first order derivative solution to be closer to the original second-order one, reducing the complexity of the overall algorithm.

The new objective function becomes:

$$\max_{\theta} E_t [r_t(\theta) A_t] - \beta \hat{E}_t [\text{KL}(\pi_{\theta_t}, \pi_{\theta_{t+1}})] \quad (2.2)$$

Where  $\beta$  controls the weight of the penalty and can be adaptively shrink or expanded based on the previous KL-divergence.

**Clipping Objective** In their paper, [Schulman et al., 2017], the authors refer to a clipped surrogate objective function:

$$L^{clip}(\theta) = E [\min(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) A_t)] \quad (2.3)$$

Where the advantage function is clipped if  $r_t(\theta)$  falls outside the range of values given by  $[1 - \epsilon, 1 + \epsilon]$ . This step is introduced in order to keep the difference between the new and old policies bounded and not higher than  $\epsilon$ .

**Scalable Distributed Deep-RL** The aforementioned algorithms work in a synchronous way, that is the the action sampling and the training phase are consecutive, see Figure 2.7.

In their work [Espeholt et al., 2018] focus on the building of a distributed architecture which is known as IMPortance weighted Actor-Learner Architecture (IMPALA), which allows to maximize data throughput in the model. This novel design grants the collection of experiences by multiple agents in a decentralized fashion, see Figure 2.8, which are then passed to a centralized learner to compute the gradient.

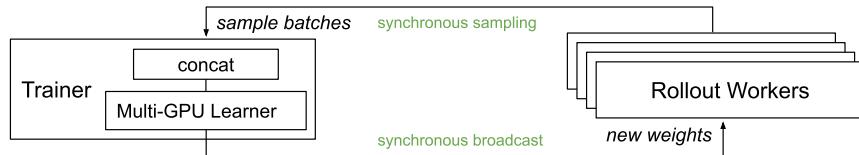


Figure 2.7. PPO architecture <sup>16</sup>

<sup>16</sup>Image taken from the [ray library](#).



Figure 2.8. IMPALA architecture <sup>16</sup>

## 2.3 Environment

In this section a detailed study of the implemented game will be pursued. During the entire section the following notation will be used:

- $n$ : number of players
- $SL$ : length of the signal.
- $SR$ : range of the signal.
- $w$ : number of werewolves

### 2.3.1 Tool-kits and Libraries

Before diving into the technical aspects of the game implementation, we need to reference the libraries and tool-kits that made the project possible.

**OpenAI Gym** For many reinforcement learning setups, the well known *OpenAi Gym* toolkit [Brockman et al., 2016] is used. This tools provide a set of stable benchmarks with common interfaces to implement games logic. Each game is built by implementing the abstract *Environment* class, more specifically the *reset* and *step* methods.

Since its release in 2016, the number of environment has increased drastically; as per today, May 2020, their [github page](#) counts more than 780 different environments.

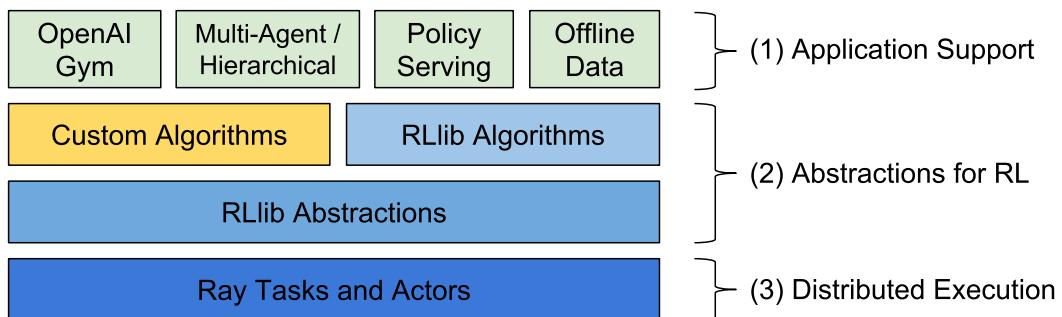


Figure 2.9. *Rllib* stack. You can see the usage of the *OpenAi gym* environment as well as the distributed *Ray* framework. Taken from [their documentation](#).

**Ray: Rllib** While the *OpenAi Gym* toolkit provides a common ground to build the game logic, the *Ray* package [Moritz et al., 2018] allows for a fast distributed system to run machine learning experiments. Precisely the *Rllib* library [Liang et al., 2017] build a high scalable, unified API for the implementation of RL algorithms. *Ray* is highly modular, splitting each aspect of its implementation in different modules to allow fast development of complex structures; the complete stack is shown in [Figure 2.9](#).

### 2.3.2 Spaces

Dealing with RL implies the usage of an action and state set; the latter are referred as action space and observation space in programming. These spaces are necessary for learning as they define a range of possibilities for what the agents may expect from the environment (*observations*) and how they can influence the game (*actions*). In general, spaces have some common proprieties depending on the field of application. These proprieties regards the range and the length of possible values the space can have; the range refers to the values being *Discrete* or *Continuous* while the length is linked with the output's *dimension*.

**Discrete vs Continuous spaces** In the field of mathematics a variable can be either continuous or discrete. *Continuous* variables are allowed to take on an uncountable set of number in a particular range, practically speaking a continuous space is define by two values, a upper and a lower bound, and the variable can assume any value in that range, e.g. a non empty range of real numbers. On the other hand a *discrete* space is defined by a finite set of possible values; common examples are variables that must be integers, non-negative integers, positive integers, or only the integers 0 and 1.

**Dimensions** A space is characterize by a dimension  $D$  which determine the shape of the output vector. The most used dimensions in RL are:

- Scalars :  $D = 1$ .
- Matrices :  $D = 2$ .
- Tensors :  $D = 3$

### Action Space (AS)

As previously mentioned the action space defines the actions each player can take in order to shape the environment around it. A proper action space should balance between conciseness and specificity, e.g. an extensive *AS* may allows for multiple strategies to develop but can slow significantly the learning curve, in some extreme cases it can get stuck in a local minimum. On the other hand a condensed *AC* may not give the player sufficient maneuver to change the environment and develop new strategies.

In the werewolf environment the action space is divided into two parts: target and signal.

**Target** The target  $t$  is a discrete value in range  $t \in [0, n - 1]$ , where  $n$  is the number of players. Its intended usage is to allow players to vote for other players during the game. The range of possible values never change during the execution <sup>17</sup>, instead illegal actions, such as voting for a dead players, are filtered out later in the model<sup>18</sup>.

**Signal** To implement the communication between agents, a signal is used. This action space can be set by the user before the start of the training and defines scope of communication between agents. The variables signal length  $SL \in (0, \infty)$  and signal range  $SR \in (2, N)$  are used in order to define the valid space for communication. It is worth noticing that the upper bound for the  $SR$  is set to the number of players  $n$ ; this must be the case since the *OpenAi Gym* toolbox does not allow neither multiple action spaces nor  $AS$  with different ranges of possible values. The feasible range is later forced on the output by a filtering passage done in the model.

**Complete action space** The final action space is defined either by a *gym.spaces.Discrete* type, when there is no communication ( $SL = 0$ ), or by a *gym.spaces.MultiDiscrete* type where the length is determine by the  $SL$  variable.

### Observation Space

The observation space characterize what the agents perceive in the environment. Again, it is important to define a concise space which both describes the game accurately and does not confuse the player with meaningless information.

**Phase** To represent the four different phases of the game, a *gym.spaces.Discrete* type is used. The phases will be studied later in the [Subsection 2.3.4](#) section.

**Day** A *gym.spaces.Discrete* type is used to represent the day. Notice that the day counter starts with the value zero and is incremented by one each time the phase counter restarts. An upper bound for the possible maximum day is set to 10 <sup>19</sup>, which is enough for 21 players<sup>20</sup>. Setting the maximum number of days reduces the one hot encoder representation to a minimal length.

**Status Map** The players must be informed of the theirs and others statuses, for this a *spaces.MultiBinary* type is needed. This array maps a boolean value to an agent index, effectively informing the players if agent  $i$  is alive = 1 or dead = 0.

**Own id** As will be explained later in [Subsection 2.3.4](#), at the start of each game the agents' ids change. To help a player keep track of his new position in the game a *gym.spaces.Discrete* has been implemented. This integer in range  $[0, N]$  stays static trough out the entire play and changes at the start of a new match.

---

<sup>17</sup>Indeed a variable action space remains an open research problem in the Deep RL community  
<sup>18</sup>See [paragraph 2.4.1](#).

<sup>19</sup>It is trivial that the maximum number of days can be programmatically set to the the number of players minus the number of wolves.

<sup>20</sup>On average, 21 players spend 8.55 days to complete a game.

**Targets** Communication implies both a sender, the action space, and a receiver. The target vector stays constant on dimension during the entire execution, thus it may be tempting to use the *gym.spaces.MultiDiscrete* space type, but the latter does not allow the vector to contain padded (-1) values, so a *gym.spaces.Box* is necessary. The usage of the padded value serves both for the start of each game, when there are no observations, and to mask the wolves voting from the villagers.

**Signal** The signal observation follows the same patterns as the one in the action space. If the length of such signal is zero then it is not inserted in the observations. On the other hand, for  $SL > 0$ , the signal is defined as a *gym.spaces.Box* of dimension  $S^{N \times SL} \in [-1, SR - 1]$ .

**Dict** Finally these spaces are condensed into a *gym.spaces.Dict* to be easily handled by later computation.

**Embeddings** It is worth studying how the length of the observation vector ( $OV$ ) is influenced by its components. The length of such vector is given by the sum of:

- Maximum Days: 10 by default.
- Phase : 4.
- Status Map:  $N$
- Own id:  $N$
- Targets:  $N$
- Signal:  $N \cdot SL$

Which results in the following formula

$$\|OV\| = N \cdot (3 + SL) + 14.$$

If you consider a game with  $N = 21$  players and a  $SL = 1$  you get  $\|OV\| = 56$  elements, for  $SL = 4$  the space increases to  $\|OV\| = 119$ . Indeed the action space size has a parabolic trend as shown in [Figure 2.10](#).

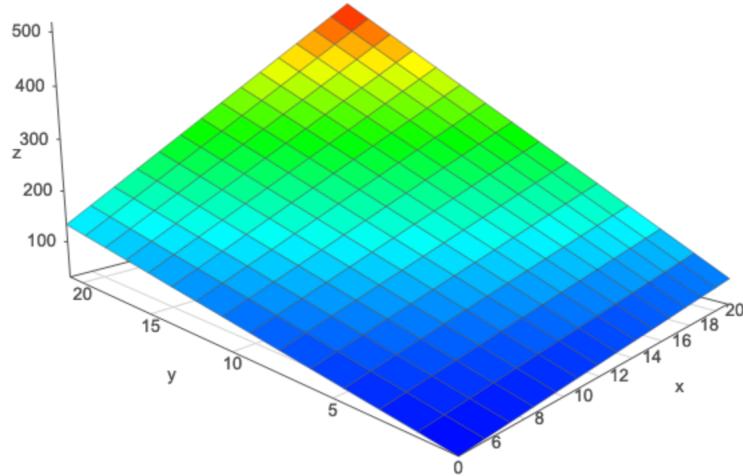
To reduce the dimensionality of such vector an embedding layer made of a single FCN is used. As seen previously, a FCN takes an arbitrary number of inputs and generate a given number of outputs, in this case the outputs are exactly  $N \cdot (1 + SL)$ . Using this network we reduce the dimensional of the output vector as well as grouping similar observation together.

The interested reader is referred to [[Kempf et al., 2006](#)].

### 2.3.3 Rewards

The rewards, or penalties, are what pushes the agents to interact with the environment. They are the core of the environment and determine how the players interact, learn and develop new strategies; the main goal of an agent is to take actions that will maximize the expected reward.

In the environment there are five conditions which determine if an agent is punished or rewarded.



**Figure 2.10.** Observation vector trend:  $x=N$ ,  $y=SL$ ,  $z=OV$

**Day** At the end of each day every agent is penalized by a small factor,  $-1$ . The goal of this penalization is to train the agents to quickly win the game, trying to develop new policies to win faster. It is worth noticing that this penalization should be kept to zero at the start of the execution to let the players focus on the game's rules rather than on the game's speed.

**Death** Each time a player dies, it takes a  $-5$  penalty. While dying can be a strategy in some matches, this behavior needs to be controlled to avoid high number of suicides.

**Target Accord** To keep the voting system to become another form of communication, the agents are penalized when they voted for someone that later on will not be killed. Since execution depends on the coordination and communication of the required group of agents, this rewards is aimed to encourage a cooperative behavior. An example follows in a four players game:

- Agent 1 votes for agent 3, agent 2 votes for agent 1, 3 votes for 1, 4 votes for 2.
- Agent 1 is executed because has more votes.
- Agent 4 is penalized.

**Win/Lost** Until now the rewards were assigned per agent. At the end of a match each group, either werewolves or villagers, is rewarded or penalized by a factor  $\pm 25$ . It is important to keep this value much higher than the others in order to keep the agents focus on winning the game rather than not dying or agreeing on the targets.

### 2.3.4 Workflow

In the following section the workflow of the environment will be analyzed in detailed. The reader is referred to [Figure 2.11](#) for the flow diagram.

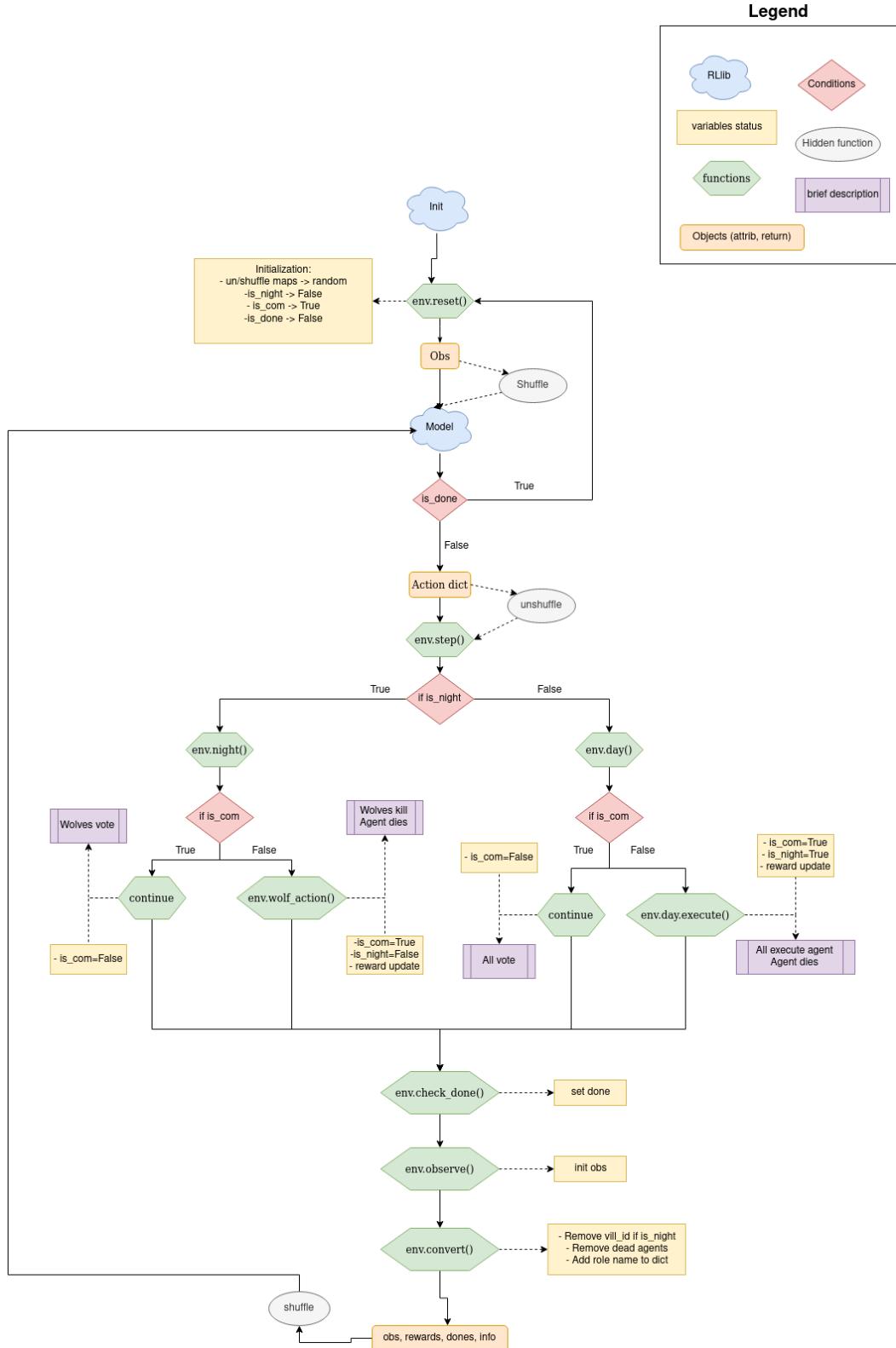


Figure 2.11. The environment workflow as a directed graph.

To determine the flow of the program three boolean flags are used:

- *is\_night*: determine the split between night and day.
- *is\_com*: used for to differentiate the communication phase from the execution one.
- *is\_done*: becomes true only when one of the two groups has won.

**Agent indexing** For implementation reasons, the agents' roles are fixed. This helps the environment to keep track of the rewards and spaces for each player with the usage of a trivial indexing list. Each player is given an unique index in the list <sup>21</sup> which will not change through out the entire execution.

**Reset** The reset method effectively starts the game, initiating most of the variables in the environment class. This method is called every time when the previous game has ended or when the program is ran for the first time.

During this call every attribute is initialized to its original value and the observations are padded to the empty value (-1).

The initial flag' values follows:

- *is\_night*: True.
- *is\_com*: True.
- *is\_done*: False.

**Phases** There are a total of four phases handled by two boolean flags.

The fist flag, *is\_night*, divides night from day while the second one, *is\_com*, splits the communication phase from the execution one. This generate a total of four distinct phases handled by the environment.

The implementation of a communication system requires two independent phases, one for the night and one for day. During this phases the agents are not penalized for non accordant target [2.3.3](#).

The phases are the following:

0. Phase : night, communication.
1. Phase : night, execution.
2. Phase : day, communication.
3. Phase : day, execution.

---

<sup>21</sup>The indices are given sequentially where werewolves are first.

**Shuffling** As previously mentioned, each agent is given unique, static id. This introduces a flaw in the game logic because, for sequential matches, the agents can learn the other players' roles , thus effectively cheating.

To prevent this behaviour, at the starts of each match, the agents' ids are randomly shuffled. This shuffling, and consequent unshuffling, takes place at the very start and end of the environment implementation, to keep the logic clean.

### 2.3.5 Metrics

To measure the changes in the agent behavior the following normalized metrics are logged into Tensorboard.

**Suicide** This is the number of times an agent votes for itself during an execution phase. Since werewolves are not allowed to vote for each other during night, this metric regards exclusively the day execution phase.

At the onset of the game, voting for oneself may be a good tactic to deceive other players. On the other hand, a dead player is forced to watch passively the game as it develops, hoping its death served a purpose.

**Wins** Both werewolves and villagers' wins are plotted in the normalized range of values.

**Average days** Another important factor is the total number of days per match. At the start, since the wolves are statistically more likely to win, the number of days roughly equals  $N - (W) \cdot 2$ , but we expect it to increase as the training progresses.

**Accord** Finally the normalized value of the target accordance is plotted. This value represent, on average, the percentage of agents that vote for the same target during the two execution phases.

### 2.3.6 Parametric Action Wrapper

The *Rllib* framework allows a *gym* environment to be wrapped around by a second, diverse environment. This allows the implementation of a hierarchical network of environment, each having its own specific goal.

As a keen reader must have noticed, there are no rules that forbid the werewolves to vote for each other or for any player to vote dead ones. At first this problem was tackled by the introduction of new penalties, but this approach slowed down the training significantly.

Another way to prevent this rule braking action to occur is to filter out the agent's output in the model <sup>22</sup> with a mask. The application of such mask requires the observations to be in a sequential, defined form, such as a an array.

---

<sup>22</sup>See paragraph 2.4.1

**Vector Observation Space** As will be later explained, the model cannot receive as input a dictionary, thus the observations need to be converted to a location invariant array. The location invariance is of fundamental importance, since the different observation spaces hold different information and the model needs to differentiate between them.

Since the *gym.spaces.Dict* holds a python Ordered Dictionary, the location invariance can be preserved simply by accessing the keys in an ordered fashion. To convert the different spaces into an array the second type of initialization for the *gym.spaces.Box* is used. In this type of initialization two vector of *highs* and *lows* defining the upper and lower bounds of each element are used.

**Masks** Contemporaneously to the observation space conversion, the mask is initialized, consisting of two independent boolean masks stacked together.

The *action mask* filters out invalid ids from the action output of the model. This ids are either dead player or other werewolves during phase 1.

The *signal mask* is a static boolean mask. Its aim is to allow only signal in range *SR* to be outputted, filtering every other value. For this reason this mask does not change during the entire execution.

### 2.3.7 Evaluation Wrapper

To keep the original environment clean, another wrapper was built on top of the previous one. The aim of the latter is to evaluate the learning, logging the matches and the metrics.

### 2.3.8 Example run

Finally an example run is presented in [Figure 2.12](#).

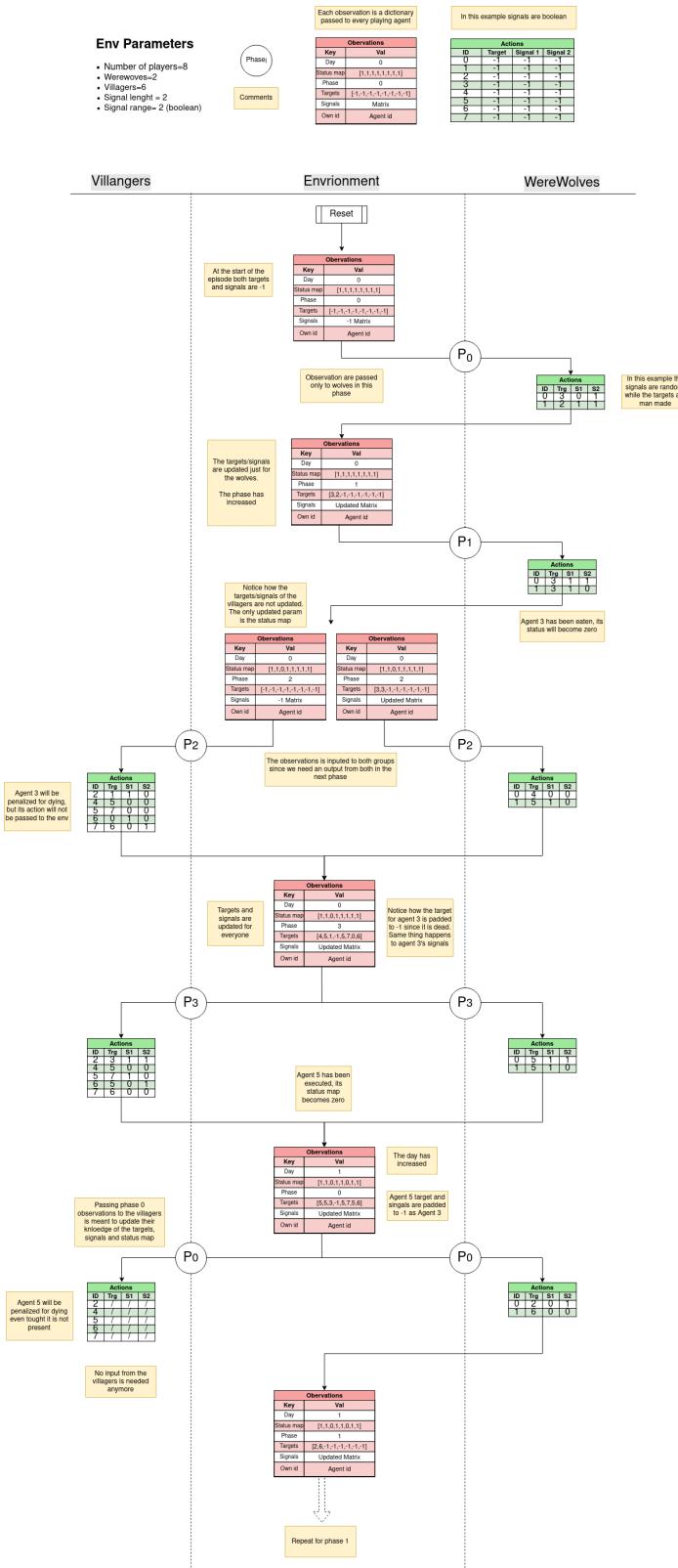


Figure 2.12. Example run

## 2.4 Policies

As discussed in [Section 2.2](#), an agent's policy defines how the learning should progress. In this environment there are two kind of policies: *trainable* policies use custom algorithms to collect experience and learn to maximize the reward; static policies are hard-coded behaviors which are used in order to guarantee a fixed baseline trough out the evaluation.

### 2.4.1 Model policies

The model is a direct sub-classing of the *Tensorflow*, [Abadi et al., 2015], version of the *Keras*, [Chollet et al., 2015], abstract network.

It is a fully connected network with  $256 \times 256$  hidden layers and the default activation function, *tanh*.

Moreover it includes an LSTM layer with a cell size of 256.

**Parametric action model** As mentioned previously, the model takes as input the observation space and filters out invalid actions. This filtering is computed by applying an negative infinite mask on the logits, i.e. the output of the FCN. Since the logits represents the probability to choose a specific output, an infinite negative mask will undoubtedly discard them.

### 2.4.2 Static

Static policies are reserved to the werewolves agents; their aim is to allow a baseline evaluation of the villager learning. Since wolves are more likely to win in a completely random environment, the application of such policies is enough to prove the development of new strategies for the villagers if the winning rates are to change significantly.

The game' rules are hand-forced into this policies, so no wolf agent will ever vote for either a dead player or another wolf. Moreover these policies work on a batch of agents, that is they receive as input a list of observation, one for each wolf, and expect as output a list of action. The upside of this methods is that the policy can access every werewolf observation at the same time, effectively allowing a hard-coded cooperation behaviour.

**Random Target** As mentioned, this policy simply chooses a random non dead player among the villagers during the execution phase.

**Random Target Unite** Slightly different from the previous one, this policy targets the same player for every wolf, both during day and night execution; this allows the werewolf to dominate the day execution phase with random villagers. During the communication phase the wolves output random targets; this tactic should keep them safe from being discovered, while simultaneously strike at the first turn.

**Revenge Target** This policy presents a more sophisticated behavior intended to test trained agents.

This policy uses a preference list in which priority targets are kept. This list is kept updated during the entire match, adding villagers who voted for wolves and removing dead players.

At execution phase, the wolves will act in one of two ways depending on the values present in the list:

- Apply the random target policy.
- Vote for a villager who targeted a wolf in the previous runs.

It is trivial to see that some kind of unite behavior may emerge if only one player is present in the preference kill list . The specifications are presented in [algorithm 1](#).

---

#### Algorithm 1: Revenge Target

---

```

Input: obs, prefList
Output: actions, prefList
Data: targets
update prefList;
if prefList is empty then
| define random targets;
else
| pick random targets from prefList;
return targets,prefList;
```

---

### 2.4.3 Trainable

As previously mentioned, *Ray* is built so to have a modular design; this can be very well seen in its implementations of the *Trainable* and *Policy* objects. A Policy is defined by ten attributes, but the most important one is the loss function which will be explained in detail in the following paragraphs <sup>23</sup>.

In the following the Proximal Policy Objective will be analyzed, for a general introduction the the algorithm please referrer to [2.2.4](#).

#### Loss function

As mentioned in the previous section, PPO uses a surrogate loss function to avoid serious drops in performance; more specifically the loss is used to keep the difference between the old and the new policy within a safe range.

Such function can generate from either [2.2](#), [2.3](#) or a combination of the two.

---

<sup>23</sup>For a complete insight on the implementation of the latter class, the reader is referred to the [GitHub page](#).

**Value loss** Since the neural network architecture is positioned in between the policy and the value function, the loss function must take into account the contribution given by the latter, thus an additional error term must be included:

$$L^{CLIP+VF} = E_t \left[ L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) \right]$$

Where  $L_t^{VF}(\theta) = (V_\theta(s_t) - V_t^{targ})^2$  is the squared-error loss between the value function  $V_\theta$  at state  $s_t$  and the target function,  $c_1$  is its coefficient. This addition is of fundamental importance to estimate the critic loss, that is the how well the model is able to predict the value of each state.

**Entropy** An additional term is introduced to regularize the total loss:

$$L^{CLIP+VF+S} = E_t \left[ L_t^{CLIP+VS}(\theta) + c_2 S[\pi_\theta](s_t) \right]$$

The entropy coefficient is maximized when all the policies have equal probability to be chosen, that is when the agent is acting at random. Adding such value to the loss function incentivizes the training algorithm to minimize the entropy value, thus avoiding the premature convergence of one action probability dominating the policy and preventing exploration <sup>24</sup>. The term  $c_2$  scales the value of the entropy.

---

<sup>24</sup>For a complete understanding of the influence entropy has on policy optimizations, the reader is referred to [Ahmed et al., 2018].



# Chapter 3

# Results

## 3.1 Baselines

Understanding how AIs learn is a hard problem in the field of machine learning. In computer vision and specifically with convolutional neural networks (CNN), visualizing the activation layers together with the filters is a common practice to better understand the structure of the network itself.

On the other hand, for RL environments, there has been some research in the area of visual diagnostic for RL systems using CNNs to elaborate visual stimulus [Luo et al., 2018]; for emergent language the preferred way to quantitatively evaluating the collaboration between agents has been accomplished by measuring the performances of the agents themselves. In their work [Barton et al., 2018] use a convergence cross mapping (CCM), borrowed from the field of ecology, to evaluate their agents' learning.

It is trivial to see that, whichever method is preferred, there must be some kind of evaluation method for the communication system. In the following, such evaluation is estimated firstly by drawing some baselines with complete random agents, and then by comparing the latter with the results yield by the training .

Such baselines refer to the probability that complete random agents have to win the game; the reader is referred to [Table 3.1](#) for the mean squared error (MSE) between these outcomes.

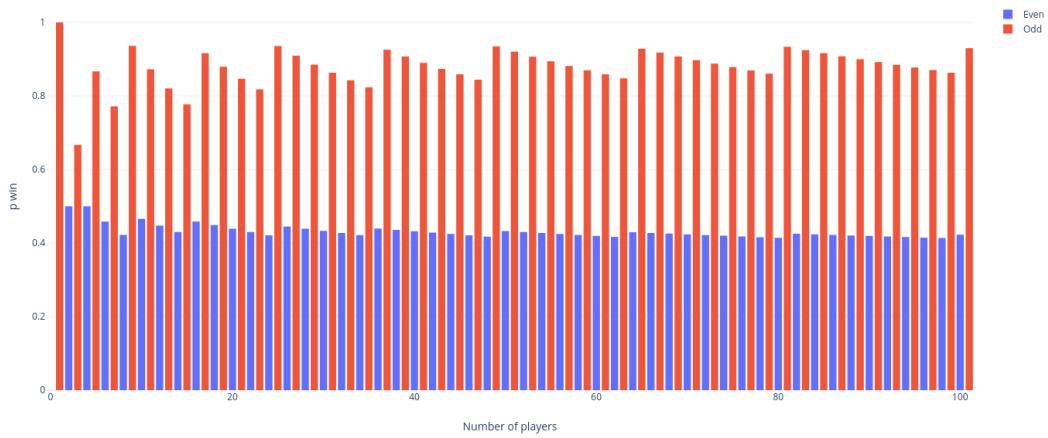
**Table 3.1.** MSE between werewolves' win ratios for different policies; the value is shown in percentage.  $T$  stands for theoretical and  $P$  for practical.

	T-random	T-Unite	P-random	P-unite	P-revenge	Tree
T-random	0	16.956	7.247	16.967	7.798	7.41
T-Unite	-	0	3.168	$1.2e^{-6}$	3.553	2.45
P-random	-	-	0	3.171	1.732	0.61
P-unite	-	-	-	0	3.556	2.46
P-revenge	-	-	-	-	0	0.76
Tree	-	-	-	-	-	0

### 3.1.1 Theoretical

In this part, the focus will be on estimating the wolves' winning rate with a closed form formula, rather than experimenting with the environment.

**Random Policy** As mentioned in [Section 2.1](#), [Migdał, 2010] found the close formula solution for the werewolves' winning probability, [Equation 2.1](#). Such formula can be used to determine the similarity between the theoretical setup and the practical one; in particular we wish to see a similar behavior to the one presented in [Figure 2.1](#). An additional plot is presented in [Figure 3.1](#) to get the reader familiar with the trend before presenting the results obtained in the next section.



**Figure 3.1.** Theoretical winning rate

**Unite** There are no formal studies which can guide our study of the unite policy.. Having performed the same amount of experiments as for the previous case, the results where unvarying; indeed, with the unite policy the winning probability for the werewolves stays constant and equal to 100%. Since the computation time does not allow for the number of player to rise above  $n = 101$ , a statistical study can be undertaken to check whenever these findings are reasonable.

The number of werewolves depends on the number of player with the following proportion:

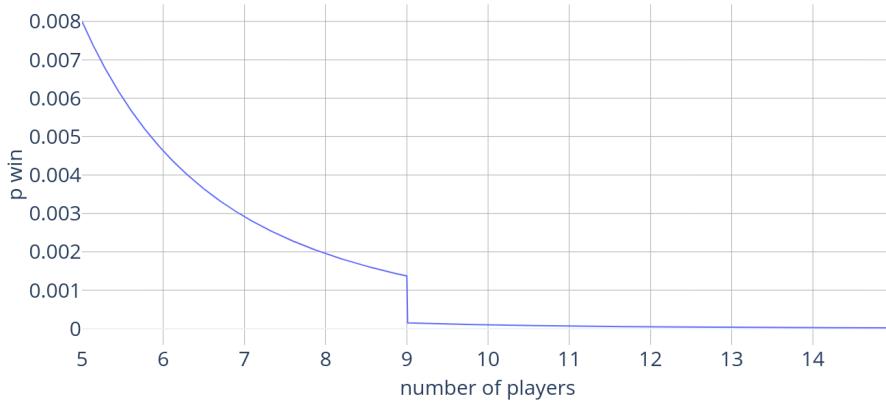
$$w = \lfloor \sqrt{n} \rfloor$$

Since the werewolves always agree on the target, there are  $w$  same targets. On the other hand,  $v = n - w$  votes are given by the villagers at random; the probability that there are more same votes from the villagers than the ones coming from the wolves is:

$$\frac{1}{n^{w+1}} = (n)^{-(\lfloor \sqrt{n} \rfloor + 1)} \quad (3.1)$$

This equation is an optimistic estimation of the real behavior since we do not consider the decreasing number of player through out the game.

It is worth noticing how the difference between this estimation and the previous one



**Figure 3.2.** Villager winning chance against unite werewolves

is not higher than 17%, [Table 3.1](#), thus this optimistic estimation could model the practical behavior well enough.

The trend for [Equation 3.1](#) is reported in [Figure 3.2](#); as can be see the function decreases exponentially to zero, starting from a mere 0.8% chance to win. There is a discontinuity present at  $n = 9$ <sup>1</sup>, where the winning probability drops to zero, this will be taken into account later in the training phase.

**Revenge policy** Finally the revenge policy is expected to mostly show the same trend as the random policy, but occasionally behave more like the unite one. This because the revenge policy heavily relies on the random procedures when the preference kill list is empty.

### 3.1.2 Practical

The generation of the practical win probabilities involved 500 episodes for each configuration. The latter was ran in parallel to allow a faster execution, but it was found to be remarkably slow anyhow; indeed the execution time is exponential in the number of players  $n$ . The parameter  $n$  ranged from a minimum of 5 player to a time-computational maximum of 101.

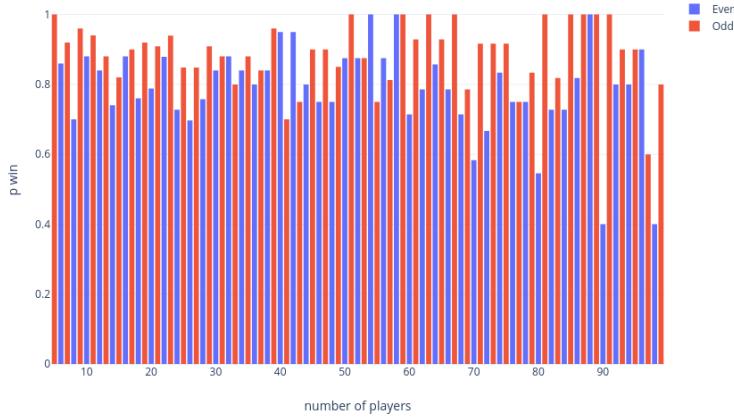
The same static policies in [Section 2.4](#) are used to guide the decision for the werewolf players; on the other hand, the villager act in a complete random way in accordance with the rules of the game,i.e. no dead man execution.

**Random Policy** This policy should mimic the same behavior studied in [Section 2.1](#), thus similar trend to [Figure 3.1](#) should emerge.

In [Figure 3.3](#) the probability ratio in the presence of random werewolf policy is presented. Each graph use a distinct color to emphasize the parity of the  $n$  parameter. It is trivial to see how the two distributions have different trends, as pointed out

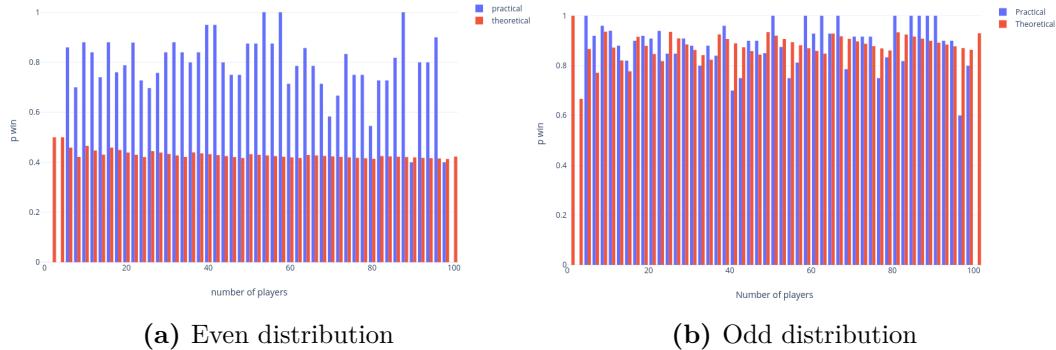
---

<sup>1</sup>When there are a sufficient number of players, the werewolves can disguise more easily in the party, increasing their probability to win.



**Figure 3.3.** Practical winning rate

by [Table 3.1](#). By graphing together same parities for both the theoretical and the practical rates, [Figure 3.4](#), the reader is able to see how the main difference comes in terms of higher probability for an even number of players [3.4a](#), while the same ratio stays constant through the odd trend. Indeed the practical trend for the odd distribution [3.4b](#) shows a similar oscillating pattern as the theoretical one, although mildly disturbed by the presence of outliers.



**Figure 3.4.** Comparison between theoretical (red) and practical (blue) probability rates for even and odd number of player

This result inspires the following section to focus on an odd number of player, namely  $n = 9$  and  $n = 21$ , for the agents' training. Finally it would be interesting to research the causes of such a different distribution for the even number of player.

**Unite policy** On the other hand the unite policy yields a constant value of 1, that is, the werewolves dominate the game for every combination of tested  $n$ . This result strongly agrees with the optimistic estimation done in the previous section; indeed the MSE between the two vectors is practically zero,  $1.2e^{-6}$ .

**Revenge policy** Finally, the revenge policy yields results close to the random one. As can be seen from [Table 3.1](#), the difference between the two is just 1.852%, thus our initial guess is indeed correct.

### 3.1.3 Expanded Tree

Finally, to have an accurate estimation on the real probabilities each game has, we study the expanded tree of possibility. Such tree is estimated by checking every possible outcome of a game with  $n$  players and counting the probability each branch yields. It is worth noticing that this kind of analysis can be performed since the scope of the project is not to mathematically give a close formula for the game trend, but instead is to study how the emergent communication can influence cooperation in a multi agent setting.

As can be seen from [Table 3.1](#), these probabilities are closer to the one estimated in the practical setting, rather than the one given by the closed form formula.

## 3.2 Nine Players

A game with nine players is relatively short.

These agents determine a simple set of actions which can be easily mapped in a table; for this reason the following section will focus only on nine players settings (9P). But before showing any results we would like to study the complete behavior of a 9P setting by inspecting its tree of possibilities.

**Expanded tree** An expanded tree, or tree of possibilities, shows every outcome of a particular process. At each possible intersection a new branch is created associated with a probability  $p$ ; in the context of the werewolf game, each branch represent an outcome where, during the execution phase, the player have killed either a villager or a werewolf.

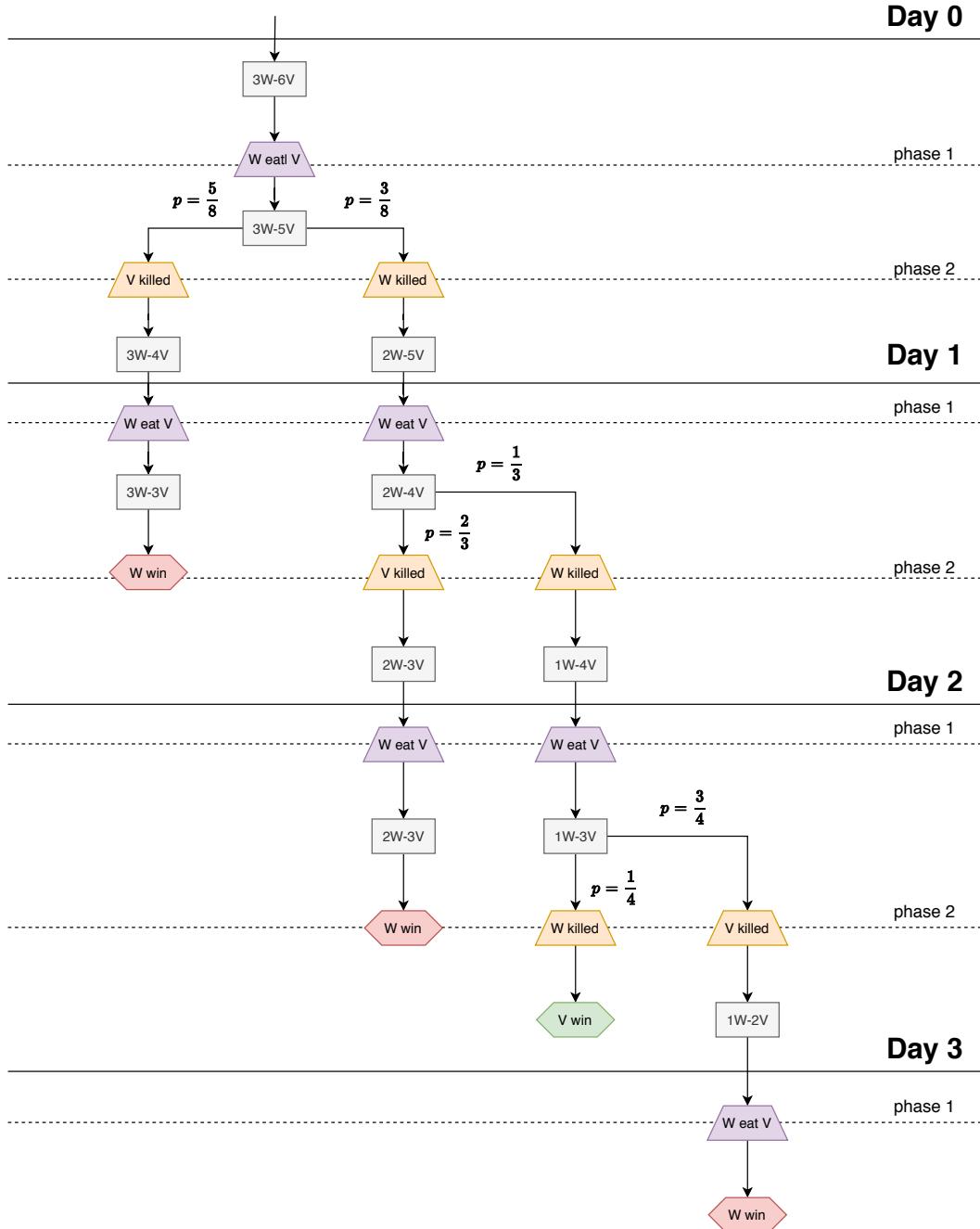
It is worth noticing that in a random 9P environment, the villagers are most likely to loose. As can be seen in the expanded tree for the game <sup>2</sup>, [Figure 3.5](#), the villagers either execute a werewolf during the first day, with probability  $p = 3/8$ , or they loose. Moreover the only chance to win is achievable with probability:

$$p = \frac{3}{8} \cdot \frac{1}{3} \cdot \frac{1}{4} = 0.03125$$

Hence they have a probability of 3.12% to win with a complete random policy.

---

<sup>2</sup>Notice how this tree has only four leaves.



**Figure 3.5.** Expanded tree for a 9 players game with 3 wolves and 6 villagers

### 3.2.1 No communication

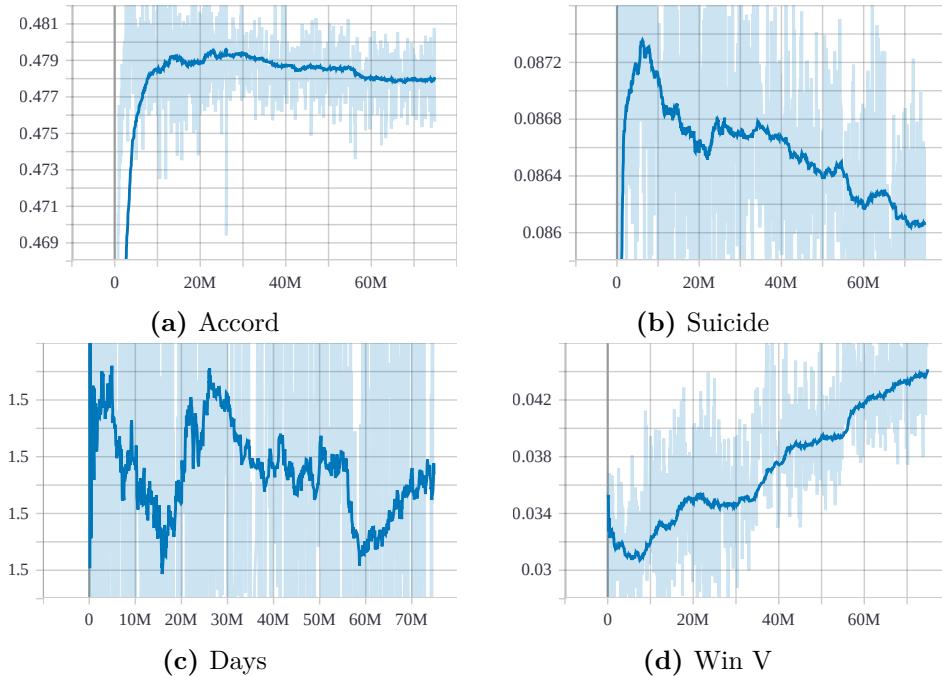
In this section the results for the training without the use of the communication will be discussed. We believed that this particular case should be discussed separately from the others, since it serves as a second baseline for the next sessions.

As mentioned in Subsection 2.3.2, the absence of communication directly maps with

a signal length of value  $SL = 0$ . The latter causes the action space to be simplified from a *MultiDiscrete* to a *Discrete* type, and the observation space to remove the signal entry all together.

### Random Policy

As shown in [Figure 3.6](#), the villagers do not show a significant learning curve. In the following a paragraph will be dedicated for each metric, the reader is referred to [Subsection 2.3.5](#) for an accurate description of the metrics.



**Figure 3.6.** Metric value for random policy with 9 players and no communication

**Accord** Being an immediate penalty, the accord sees a sharp increase of 1%. Since the villager's probability to randomly vote for the same target is at least 5%, this result is not statistically significant to show any kind of emergent cooperation between the players.

**Suicide** As mentioned in the game [Subsection 2.1.2](#), voting for oneself is often a strategy in a stand up match. In this case, committing suicide does not seem to be regarded much; indeed the value oscillates between 8.7% at the start and stabilizes on 8.6% as the learning progresses. Statistically speaking, in a totally random environment, the probability of targeting yourself is simply  $\frac{1}{N}$ , where  $N$  decreases as the number of living players reduces throughout the game. Again the slight decrease in its value does not represent a strong enough evidence for any kind of learning.

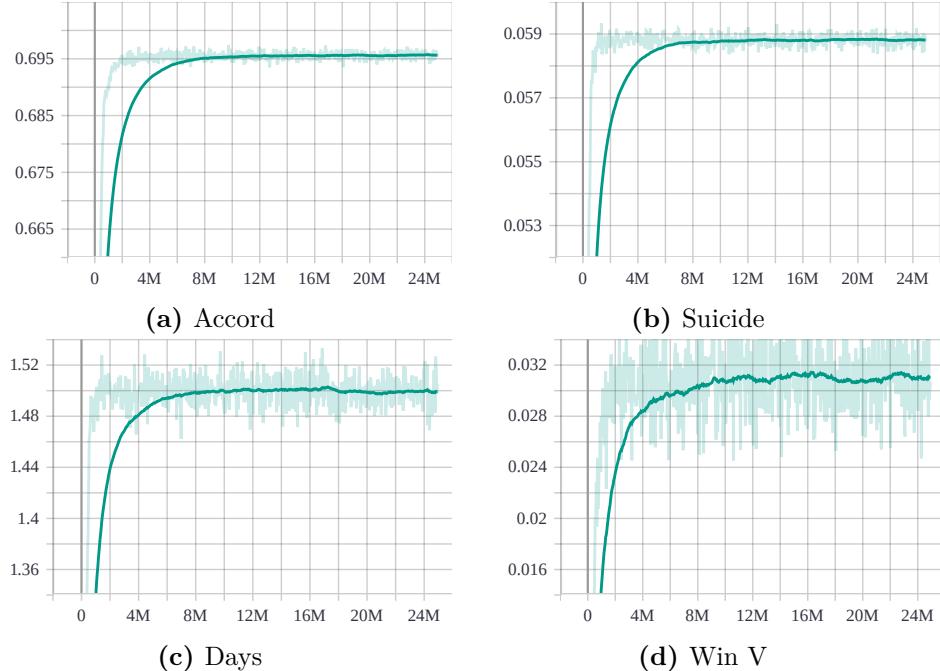
**Days** Since there is no significant learning, the number of days stays constant through out the run. As can be seen from Figure 3.6, the 1.5 value matches exactly the most likely event in the game, the first werewolf winning branch.

**Villager Win** Finally, the villager winning rate increases slightly from a value of approximately 3.2% to over 4.4%.

The main result is that the initial value is very close to the statistical guess defined in the previous section; indeed the behavior of the agents at the start of the game is purely random. After  $80M$  iterations, the likelihood to spot the werewolves during the first days increases slightly.

### Unite Policy

Again, as showed in Figure 3.7, the agents quickly learn to play, reducing the number of suicides and increasing the villager winning rate. For this training phase more steps were taken, indeed the whole execution took 63% more steps to converge.



**Figure 3.7.** Metric value for unite policy with 9 players and no communication

**Accord** Similarly as in the previous case, the accord value sees a sharp increase; starting from a higher baseline as earlier, due to the wolves voting for the same villagers, it increases sharply until the  $6M$  step, then it halts. This slight increase of approximately 3% is not very relevant.

**Suicide** This trend looks similar to the accord one; it starts from a value of 5% and increases until the  $4M$  iteration when it reaches almost 6%. The increasing is

due to the fact that the villagers are trying to maximize the accord part of their reward, effectively voting for themselves to agree with the wolves.

**Days** As before, the trend seems to be constant in all the metrics. The mean number of days spent playing slightly increases by 0.2 as the learning proceeds, until it stabilizes at  $4M$  iterations; this is in direct contrast with the suicide trend but perfectly describes the villager winning rate.

**Villager Win** Finally, the villager winning rate doubles in a short amount of time. Although the increase is slight, about 1.5%, it is still significant when it compares to the statically analysis done in the previous section; indeed, the villagers, are spotting the werewolves 20 times more often than they would without learning. This result can not be easily explained by this metrics alone; an educated guess<sup>3</sup> can be taken as follows: at the start of each game the villagers vote randomly for the same player, if the player is a wolf then the match will continue and the day counter will increase above 1; if so then the villagers try to catch the werewolves at the cost of being agreeable, if not they tend to vote alongside the werewolves to be rewarded at least 1.

### Revenge Policy

In the following, we study the case of the revenge policy for an environment without communication. The reader is reminded that this kind of policy is strictly correlated with the previous ones, since its trend is a mix of random and unite. The reader is referred to [Figure 3.8](#) for the coming paragraphs.

**Accord** It is worth noticing how this plot seems to show the initial behavior encountered in the previous policies, but it presents a hill in the range of steps going from  $20M$  to  $40M$ . This information combined with the next trend will lead to the conclusion that the revenge policy is more easily spotted than the random one. In fact, the hill is an indicator that the learning agents first try to maximize their rewards by agreeing with the werewolves, but then they tend to discard the enemy targets in favor of their own.

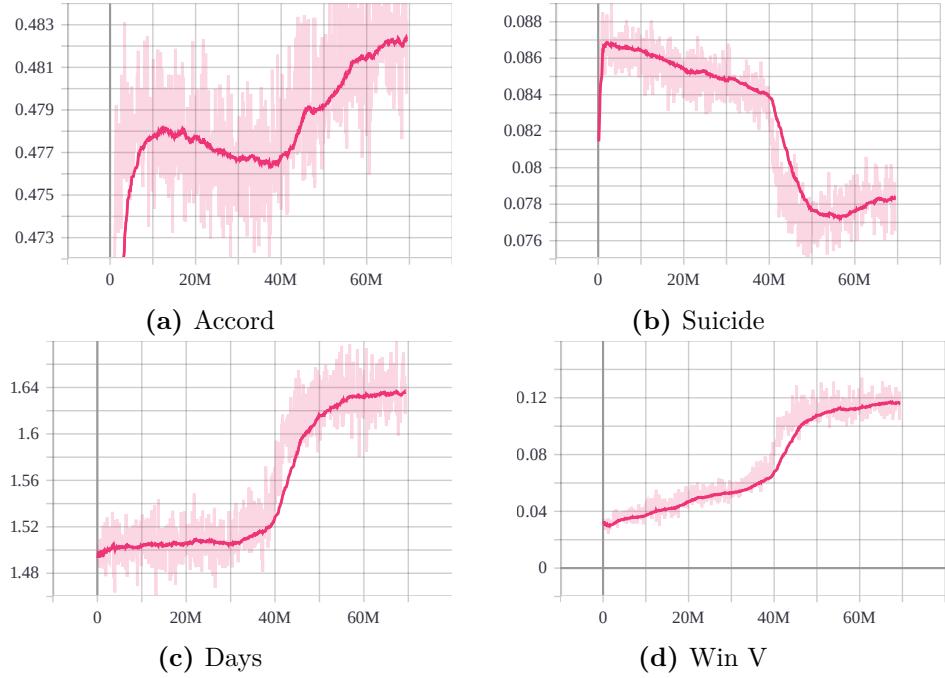
**Suicide** Even though the range is small 1%, the drop in the suicide number happens around the same time as the steep learning curve; effectively linking this trend to the aforementioned result.

**Days** the number of days increases proportionally with the villagers' learning curve.

**Villager win** The ratio with which the villagers learn increases drastically around  $40M$  steps. The value jumps from 6% to 12% in a short amount of time indicating how the villagers are able to spot the werewolves; during this interval the agents

---

<sup>3</sup>This guess is mainly coming from the match logs in which we are able to see the strategies applied by the players.



**Figure 3.8.** Metric value for revenge policy with 9 players and no communication

learn to discard the werewolves target and actively cooperate to kill them in the execution phase. This can be seen from both the Accord and Suicide trends being inversely proportional to each other; indeed the villager stop agreeing with the wolves all together, de facto not voting for themselves, and start targeting their enemy. This kind of steep learning is a behavior not seen in the previous policies, the motivation lying within this results could be the simplicity of the policy itself, i.e. the wolves can not hide behind purely random action anymore and are not strong enough to drive the majority of the votes toward a villager.

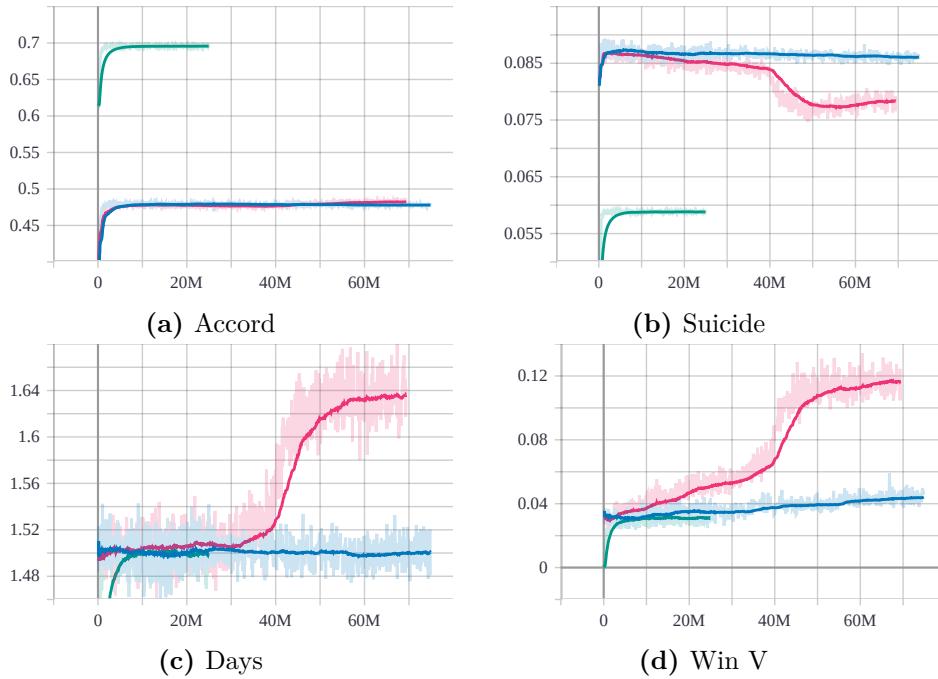
### Comparison

Finally the policy are compared in [Figure 3.9](#).

Although the unite policy has been trained for far fewer steps as the other two, it shows a constant behavior with no sign of change. As can be seen from *part (a)*, the accord trend is the same for both the revenge and the random policy and much higher for the unite one, as previously anticipated. The same inverted tendency is shown in *part (b)* where the lower value of suicides for the unite policy is given by the wolves not voting for themselves in the execution phase; moreover the lowering rate in the revenge policy can be seen starting from step  $10M$ , while the random policy does not show any sign of change.

During the first  $30M$  steps, all the policies have the same number of days, *part (c)*; but the revenge one sees an exponential increase at step  $40M$ .

Finally the main difference between the policies can be observed trough *part (d)*. Indeed the unite winning rate stays constant together with the random one, but the



**Figure 3.9.** Comparison between different policies for a 9P environment with no communication. The colors mapping are the following: blue-random, green-unite, magenta-revenge

revenge rate increases steadily by a linear factor until it reaches the turning point in which the growth assumes an exponential trend.

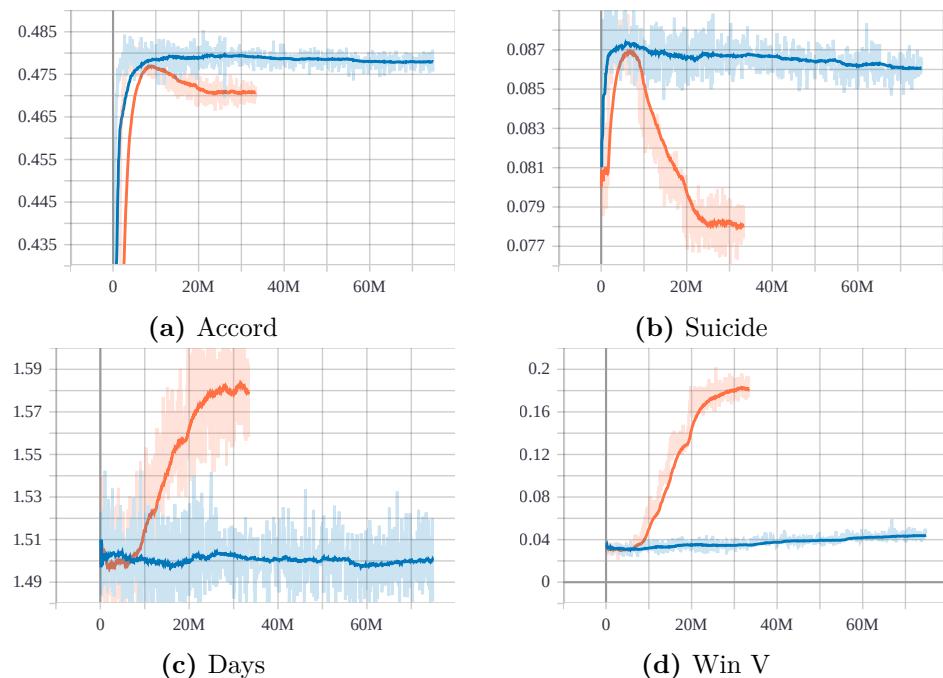
### 3.2.2 Bit communication

In the following section the bit communication will be studied together with the three different policies; this kind of communication channel sees the the signal length  $SL = 1$  and a boolean range  $SL = 2$ . Moreover the plots will show for each policy two trends, one associated with the bit communication and the other with the previous section, i.e. no communication; this is done in order to accurately spot the differences due to the introduction of a signal channel.

#### Random

As can be seen from [Figure 3.10](#), the training step greatly differ between the two run; for the no communication environment (blue) the steps are twice as many as per the bit communication case. This difference further emphasize the effect that the signal channel has on the game, indeed with just  $30M$  steps the agents are able to quickly spot the werewolves. In the following paragraphs we will study metric by metric the characteristics of each setting.

**Accord** This trend initially behaves as the previous case, approaching 4.8%, but it lowers towards 4.7% when the villagers are able to correctly distinguish their targets from the werewolves' ones.



**Figure 3.10.** Random policy with 9 players for both no communication (blue) and bit communication (orange)

**Suicide** The previous is confirmed by the sharp drop in the suicide rates shown in part (b). In fact the rate drop from 8.7% to 7.8% at the same time as the suicide rate; this shows how the coordination within the group is effectively allowing the villagers to recognize and discard the enemy votes, ceasing to target themselves.

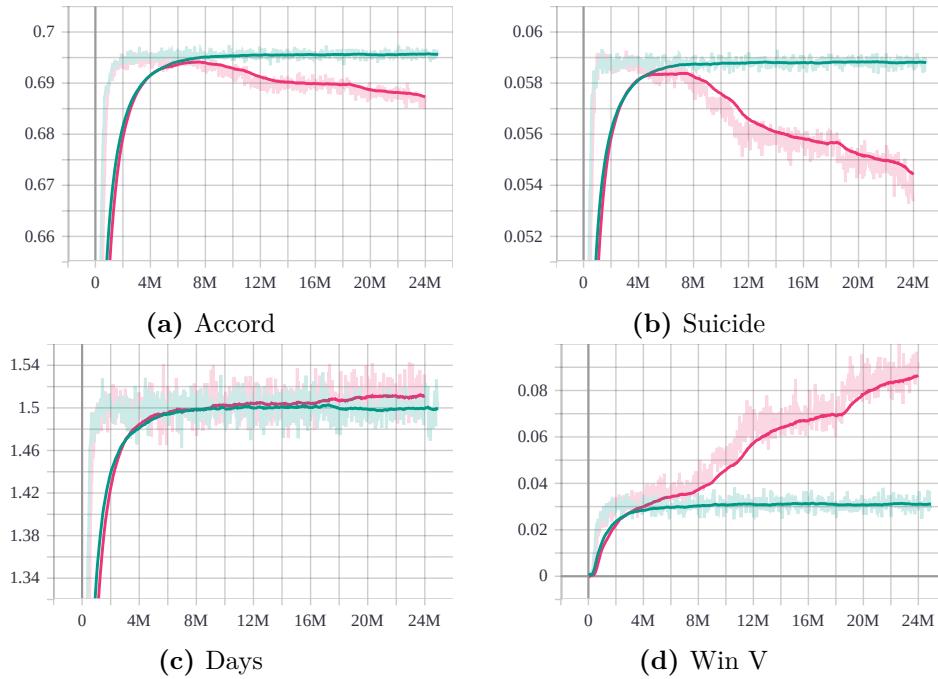
**Days** It is no surprise that the number of days increases as the players learn to spot the werewolves. Indeed this metric is always proportional to the villagers winning rate.

**Winning rate** The most evident disparity between the two environment is given by the magnitude of the villager winning rate. As a matter of fact, the increase encountered in the previous setting is nullified by the exponential growth the rate is subject to in this case. The villagers' winning reaches 20% in just  $30M$  steps; that is 4.5 times more than the previous condition and 6.5 times the theoretical winning rate.

Unite

As in the previous case, the coordination is much grater. Indeed the villagers are able to increase their winning rate by a factor of  $\times 3$ , Figure 3.11.

**Accord** The decreasing of the accord value is exactly what to be expected by the villagers when dealing with the unite policy. As can be seen from *part (a)* of Figure 3.11, the mean agreement value decreases by 1% which is not much



**Figure 3.11.** Unite policy with 9 players for both no communication (green) and bit communication (magenta)

compared with the statistical analysis, but a strong result when compared with the no communication case.

**Suicide** The suicide metric seems to confirm what previously said, that is the villagers are learning to discard the werewolves' targets and stop voting for themselves. This new strategy is only possible thanks to the introduction of a communication channel.

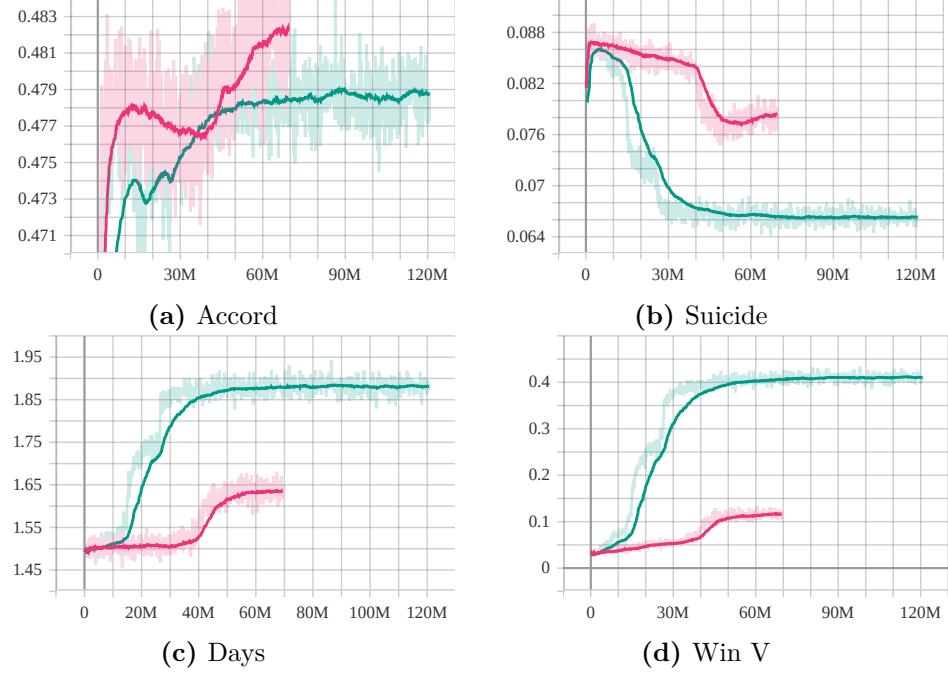
**Days** There is no surprise to see how the average day length for a match increases as the agents learn to play better.

**Winning rate** Finally the villager winning rate sees a statistically strong increase. The rate triple in a short amount of time compared to the setting without communication. Moreover, this new value is infinitely greater than the one encountered in the practical analysis in [Subsection 3.1.2](#); each and every of these results point toward an increased cooperation with the introduction of a single bit as a communication channel.

### Revenge

As seen previously, the revenge policy is the easiest to spot for a trained agent; indeed the win ratio for the no communication setting increases above any other ratio found in the other policies. It is trivial to see from [Figure 3.12](#), that the introduction of a communication channel greatly helps the training process in both

performance and speed. Even though the number of steps is greatly superior in the communication setting, the learning can be seen appearing far before the other case, and stabilizing right after.



**Figure 3.12.** Revenge policy with 9 players for both no communication (magenta) and bit communication (green)

**Accord** The accord value sees the usual sharp increase, but, instead of presenting a big hill where the agents learn not to trust the werewolves, it has a series of two small variations before converging to the final value. These variations are representative of the aforementioned learning in a more compact and efficient way thanks to the additional communication bit the players can use.

**Suicide** Again the difference between the two settings is astonishingly high. The suicide drops by 2% after a small increase where the agents try to agree with the wolves; it is worth noticing that in no other case we can observe such a significant decrease in this trend.

**Days** The average length of the match is another marker of how well the villagers are performing under these circumstances. The difference between these two settings is 0.25, starting from 1.65 and peaking at 1.86 ; referring to Figure 3.5, it is easy to see how skewed the game is toward the second day where the only villager winning outcome is.

**Winning rate** Subsequently, the winning rate is four times higher than the setting without communication. Approaching the maximum theoretical limit of 50% for a

total random game, the agents are performing ten times better than in the other environment with different policies.

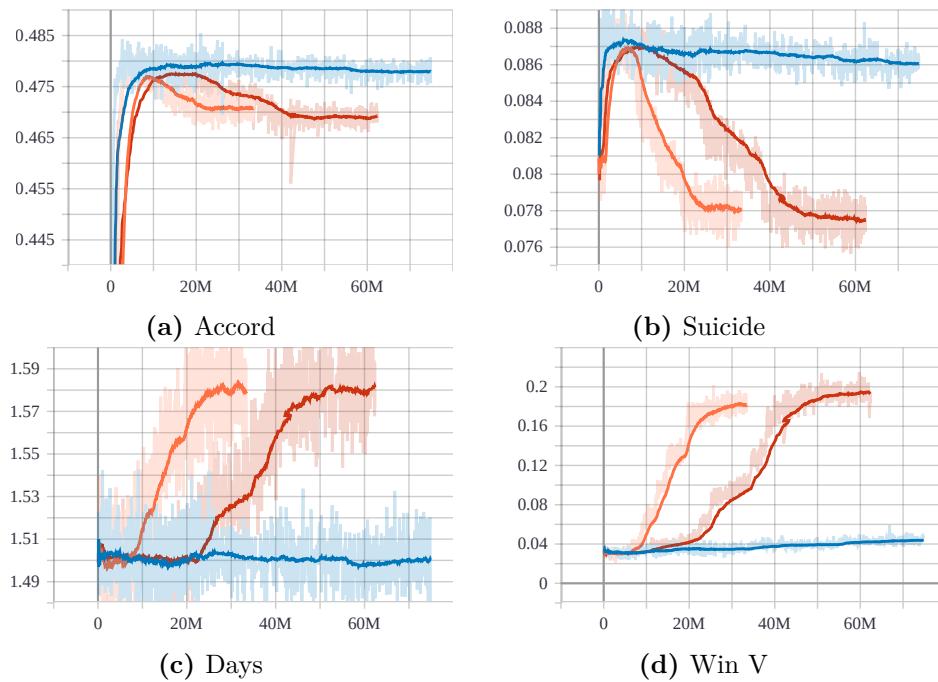
### 3.2.3 Multi channel communication

In conclusion, we study the influence on both the signal range  $SL$  and signal range  $SR$  for a 9P setting. Given the limited computation power and time, the random policy will be the only one analyzed; further researches can try to vary the aforementioned parameters for other policies too.

Since there is a limited amount of computing power, just four of the 72 possible combination will be studied.

#### 1SL-9SR

In this part, the agents are able to use an additional signal integer in range  $[0, 9]$ ; the latter is compared with the no and bit communication settings to better study their differences in [Figure 3.13](#).



**Figure 3.13.** Random policy with 9 players for both no communication (blue), bit communication (orange) and 1SL-9SR communication (red)

**Accord** The accord trend sees the usual sharp increase at the start of the training although the peak value does not reach the baseline. On the other hand, this metric presents a hill and then declines constantly upon reaching the value of 46.8%, 1% less than the baseline value. It is worth noticing that this is the first time a we register a decrease in the agreement with such a great magnitude.

**Suicide** Contrarily to the previous plot, the suicide trend does not show peculiar behaviors compared with the previous cases; indeed its value decreases constantly until it stabilizes on 7.8% with a total difference of 1%.

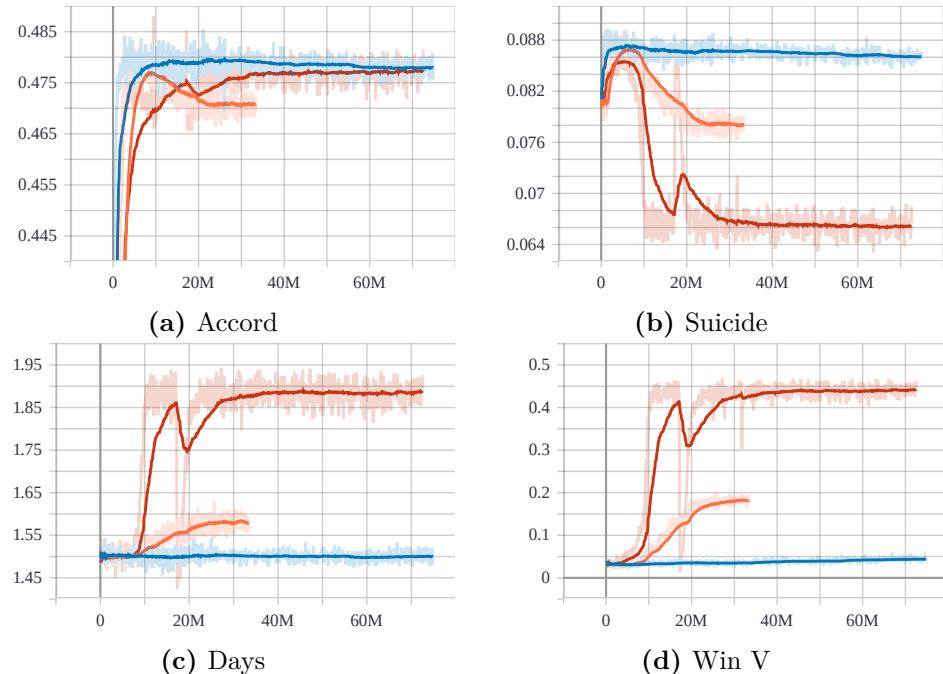
On the other hand, the current setting takes  $30M$  more steps to converge than the bit one, showing an effective slow down of the training.

**Days** The average day length does not provide additional information compared to the next trend.

**Winning rate** On the other hand the winning rate rises to a value of 20%, a remarkable outcome indicating how the villagers are five times more likely to win compared to the baseline value. Although this result is higher than the bit communication one by 1%, the time introduced for the training to converge is doubled, making the this setting worse than the one treated in the previous section.

### 9SL-2SR

The next objective is to see if the signal's length changes the training result, and if it does, to study and analyze the influence it has on the game. As can be seen from [Figure 3.14](#), the  $SL$  parameter does indeed strongly influence the behavior of the agents; in the following the examination will focus on the discontinuity present around iteration  $20M$ .



**Figure 3.14.** Random policy with 9 players for both no communication (blue), bit communication (orange) and 9SL-2SR communication (red)

**Accord** This trend does not see any kind of different tendency compared to the other ones, if not for the bump at iteration  $20M$ . This sudden drop in the level of agreement is correlated with a decreased performance from the villagers.

**Suicide** The suicide trend decreases to an unusually low value for the random policy settings; with a value of 6.6%, it find itself 2.2% lower than the baseline setting and 1.2% lower than the bit communication environment, the lowest configuration so far.

**Days** As mentioned in the previous paragraphs, the day trend is usually just an indicator of how well the villagers are learning. Indeed in this case too it does not provide any additional information.

**Winning rate** The winning rate follows an inverse proportional trend to the suicide percentage, indeed the strategy learnt at iteration  $20M$  results to be not optimal for the agents and is quickly discarded. On the other hand, the final winning rate is much higher than the one in the baseline settings, precisely it is 10 times more than the other run.

**Analysis** Since this training has such a peculiar trend, we think it might be in the best interest of the reader to explain it. By studying the other logs for the run, we noticed how the sample and train throughput dropped at around the same step range; these two parameters are the one responsible to actively count how many games are passed to the model for learning. A drop in their value correspond to a lower training batch <sup>4</sup> which results in the sudden lack of appropriate feedback. This causes the algorithm to deviate from the original trend. Practically speaking the machine is at fault rather than the process itself, but we think this kind of error is worth reporting since dealing with deep learning in general can be prone to many kinds of unexpected behaviors which require some further analysis.

## 9SL-9SR

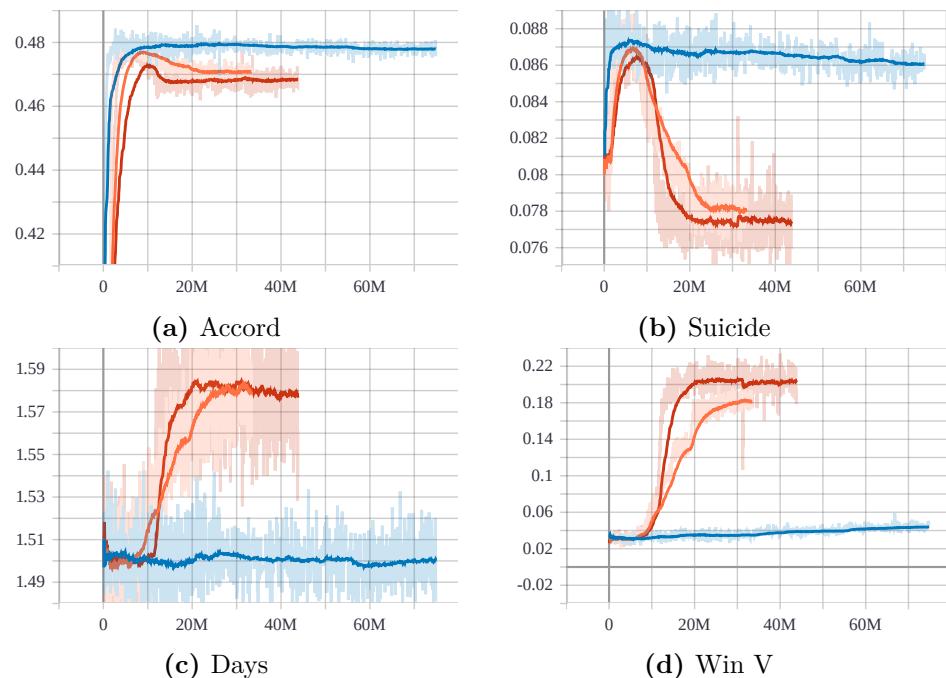
By maximizing the amount of information exchangeable between the agents the training performs highly better than the baseline, but not a good as in the previous cases.

**Accord** The agreement value increases similarly to the baseline with a slight delay of  $50k$  steps, reaching a maximum of 47%. Then it drops in a few million iterations to stabilize at 46.7%.

**Suicide** This trend behave likewise the previous one, with a delayed increase which reaches the value of 8.6% before dropping to 7.7% in just  $17M$  steps. As will be discussed later, this behavior shows an initial phase in which the agents are trying

---

<sup>4</sup>A training batch is an array of experiences which allows the learner algorithm to take as input many episodes at the same time for efficiency reasons.



**Figure 3.15.** Random policy with 9 players for both no communication (dark blue) and 9SL-9SR communication (red)

to maximize the reward portion coming from the agreement part, until they learn that going forces against the werewolves is a much better strategy.

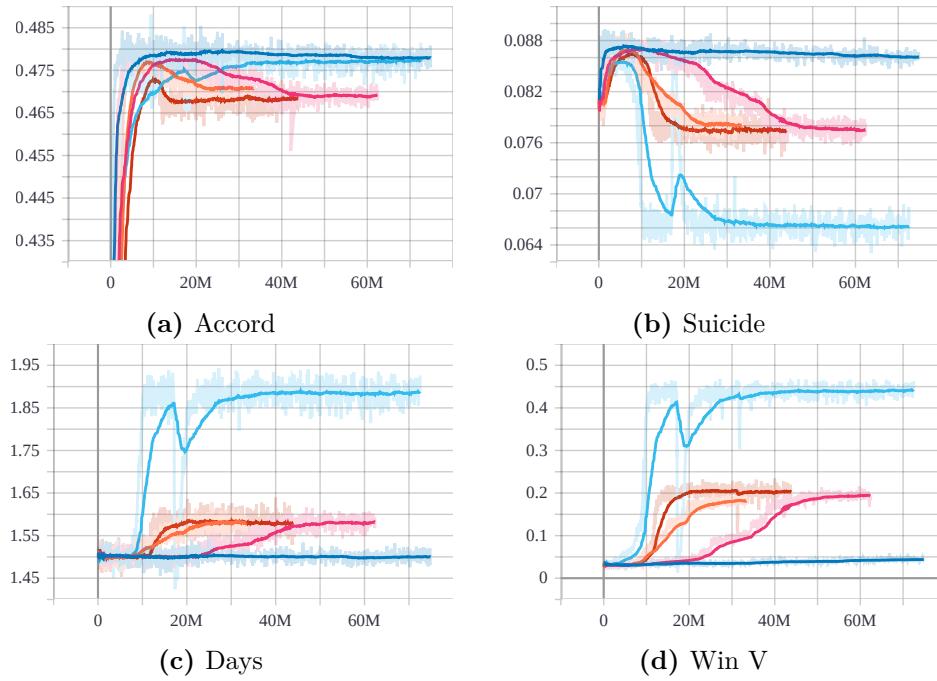
**Days** The average day length plot does not provide more information than the one presented in the next chart; indeed it shows an identical trend to the winning rate.

**Winning rate** The villager winning rate rises from the baseline value, 3.1%, to 20.5% in just  $20M$  steps. This result is not as high as the previous ones but it is faster, which may point out how there is an optimal combination of signal range and length that may yield the highest winning rate in a minimum number of iterations.

## Comparison

Figure [Figure 3.16](#) reports all the metrics for every type of communication treated so far for the random policy.

It is trivial to see how the bit communication (2SR) is doing much better than the full ranged one (9SR); indeed with the addition of just one bit the villagers are able to perform as good as the full extended communication (9SL-9SR). We can infer from the figure how extending the range of the communication leads to better performances which are constraint below a fixed point, i.e. 20%; this can bee seen from both the 9SL-9SR and 1SL-9SL results which reach the same winning rate in a different amount of iterations. In fact, for the settings where  $SR = 9$ , increasing the channel length  $SL$  has the only effect of speeding up the convergence by a factor of



**Figure 3.16.** Random policy with 9 players for the following communication types: 9SL-2SR light blue, 9SL-9SR red, 1SL-2SR orange, 1SL-9SR magenta, OSL dark blue.

circa 25% for each increase.

On the other hand the bit communication, however small the channel length might be, provides much better results. This leads to the conclusion that the two parameters dictating the change in the communication channel are not equally influential when it comes to the agent's learning; we see how the introduction of an extended signal range can be considered damaging to the training algorithm.

### 3.3 Twenty one players

Differently to what studied in the previous section, the twenty one players (21P) environment has more room for the villagers to win in a completely random setting. Indeed, by applying [Equation 2.1](#), the estimated winning ratio for the werewolves players is :

$$W(w, n) = W(4, 21) = 84.68\%$$

Which is 10% less than the previous case (93.65%). As done previously, we will first analyze the expanded tree to understand the possible outcomes as well as the probabilities tied to each one.

**Expanding Tree** Since the binary tree spanned by the combination of 21 players is too large to show, the reader is referred to [Table 3.2](#), where the possible outcomes are associated with the relative probabilities and number of leaves.

**Table 3.2.** Mapping between outcomes and probabilities. The outcome shows the number of werewolves-villagers

Outcome	Prob. %	Leaves	Total %
0-12	0.029	1	
0-10	0.143	4	
0-8	0.447	10	
0-6	1.162	20	11.62
0-4	2.819	35	
0-2	7.02	56	
1-1	21.06	56	
2-2	27.965	21	88.38
3-3	26.017	6	
4-4	26.017	1	
<b>Total</b>	<b>1.0</b>	<b>210</b>	<b>1.0</b>

As can be seen from the table, the possible distinct outcomes are 10, with a minimum of 1 leave for the 0 – 12 outcome <sup>5</sup> to a maximum of 56 for both 0 – 2 and 1 – 1. Although the number of leaves for the latter outcomes are the same, it is trivial to see how the probabilities are heavily biased towards the werewolves winning, 21% against 7%; indeed the total probability for the werewolf victory is 88.38% while the villagers are stuck at 11.62% <sup>6</sup>.

### 3.3.1 No communication

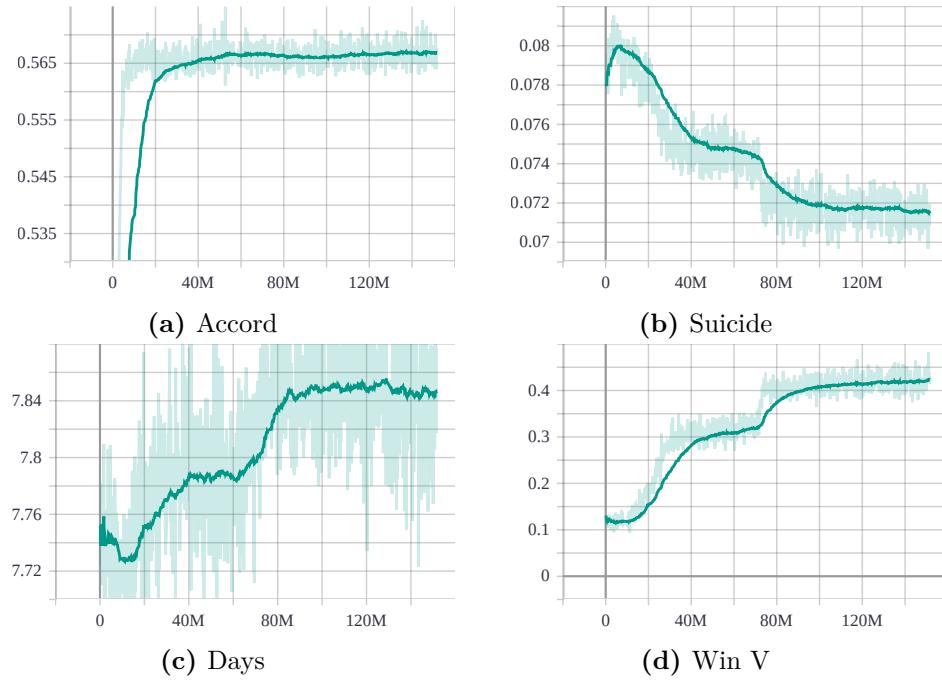
Since the computational time is much higher for the 21P setting, the following section will focus on the random policy only.

As always, the baseline environment with no communication is shown in [Figure 3.17](#); in this section we analyze the main difference between this setting and the one with nine players, [Figure 3.6](#). It is trivial to see that the main difference between the two trends is the number of iterations required for the learning to converge to a semi-fixed value, naturally the introduction of 11 more players slows the training for both the increased complexity of the game and the additional players themselves.

**Accord** With more players the mean accord percentage increases, precisely 10% more agents tend to output the same target on average. This is a trivial results since more players implies both a broader set of targets to choose from but also more agents choosing. Moreover the proportion of villager is greater in this setting

<sup>5</sup>This outcome refers to the possibility of the villagers killing all the werewolves without mistakes.

<sup>6</sup>It is worth noticing how this winning chances are different from the one given by [Equation 2.1](#). Indeed it would be worth study further the reasons why this is.



**Figure 3.17.** Metric value for random policy with 21 players and no communication

than in the previous one, 81% against 66%.

For what regards the agreement trend, there are no significant changes in relation with the 9P environment.

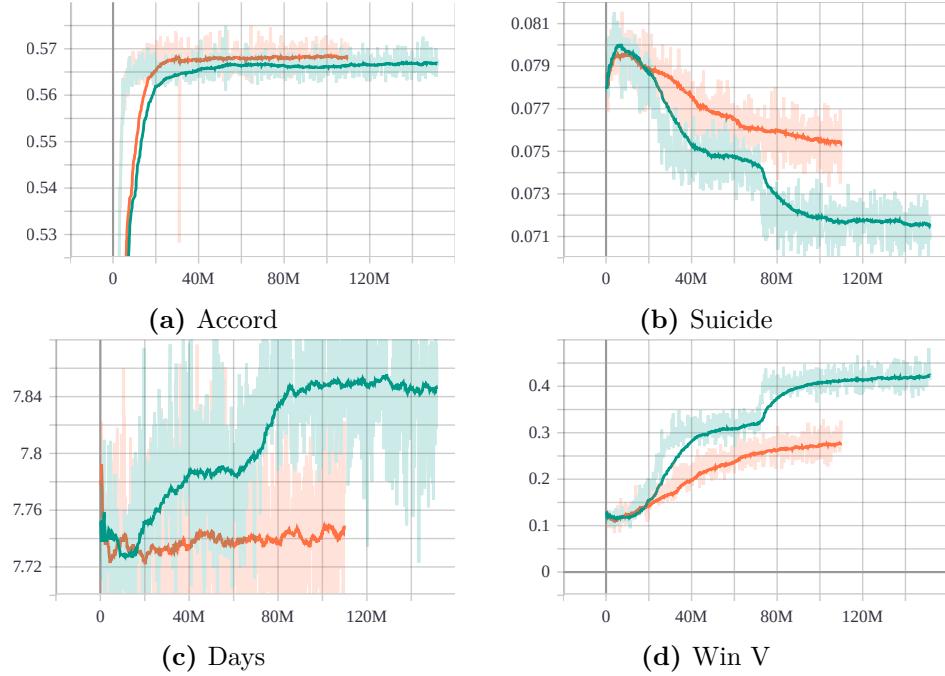
**Suicide** A broader set of possibilities to choose from decreases the chances of a non intelligent player to vote for himself; indeed the suicide rate starts from 8% compared to the 8.7% in the earlier setting. As the training progresses the suicide plot follows a similar but steeper trend to the one presented in [Figure 3.6 part \(b\)](#), in fact the plot reaches 7.1% which is proportionally equivalent to the drop encountered in the 9P setting.

**Days** The number of days does not encode additional information, it follows precisely the trend in the next paragraph.

**Winning rate** Finally, the winning rate jumps from 12%, really close to the value presented in [Table 3.2](#), to 41% at the end of the 150M step. Again, this result perfectly fits what anticipated earlier, i.e. how the villagers have more chances to win now that the match is not decided by the first action anymore. It is worth noticing that the winning rate is  $\times 3.4$  higher between the start and the end of the training while, for the 9P setting, the proportion is  $\times 1.25$ ; that is, the winning rate does not hold the same proportional invariance propriety as the suicide and accord metrics.

### 3.3.2 Bit communication

In the following, the 21P setting with bit communication is studied, [Figure 3.18](#). This setting does not reflect the findings in the 9P environment since no increase in the villagers' winning rate can be detected.



**Figure 3.18.** Metric value for random policy with 21 players with no communication (green) and bit communication (orange)

**Accord** The accord trend does not vary much from the baseline case, its value oscillates slightly around the fixed point before finally converging.

**Suicide** On the other hand, the suicide plot shows a slow convergence to a fixed value of 7.5%, 0.5% more than in the baseline trend. This behavior suggests a worse coordination than the previous one which can be attributed to a wrong exploration of the expanded tree.

**Days** The average day length does not vary much during the execution, staying steady around 7.73. It is interesting to notice how the information between this and the next plot suggests how the villagers were able to choose an early branch in the tree with a winning outcome and then stick with it.

**Winning Rate** Finally the winning rate does not increase above 30%, which is still higher than the complete random setting by a factor of  $\times 3$  but not comparable to the baseline environment.

**Discussion** From the previous paragraphs we can infer how this training instance fails to correctly explore the environment, choosing a branch with a winning outcome and sticking with it until the end. It is not clear if this outcome is mainly due to the type of communication involved or if it is just a matter of chances; same thing can be now said for the baseline instance which may be just a fortunate coincidence. On the other hand, having most of the communication settings culminating with a much higher winning rate than the baselines is indicative of the fact that the signal is indeed helping the agents exploring the possible branches better than in other cases.

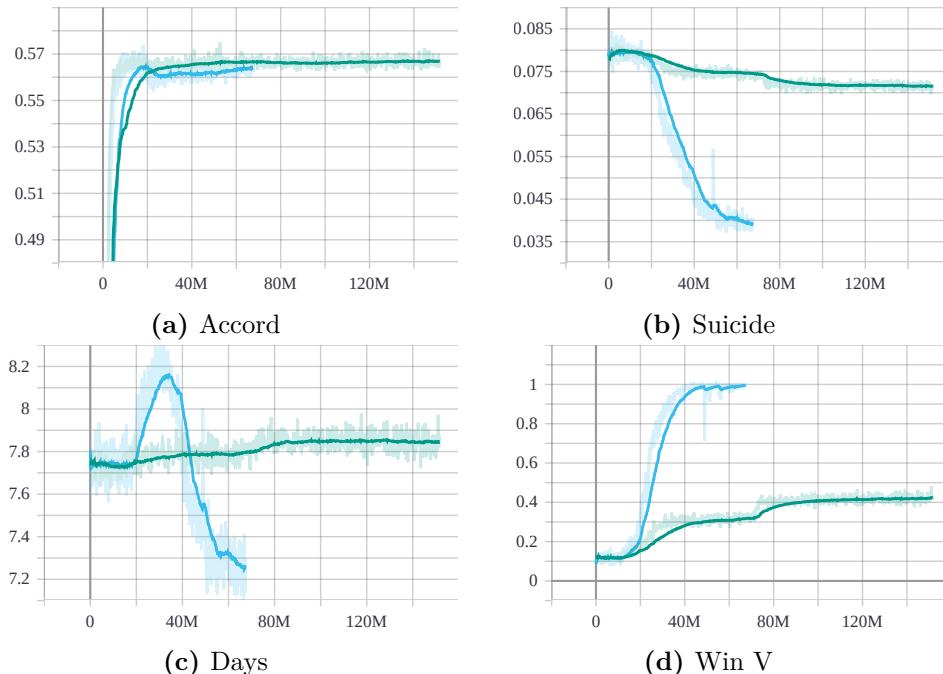
For this reasons, we think that this outcome is worth investigating because it can show how the shape of the communication signal can increase the agents' chances to find a better branch instead of just increasing the winning outcomes.

### 3.3.3 Multi channel communication

In this section, four different kind of channel configuration will be studied, each one will be compared with the baseline value and, finally, with everyone else.

#### 9SL-2SR

In this experiment we introduced an additional 8 bit for the communication. The aim is to study if there is any improvement and, if the answer is positive, measure it and try to estimate its trend. In the following paragraph the reader is referred to [Figure 3.19](#) which shows both the baseline case (gray) as well as the current one (blue)



**Figure 3.19.** Metric value for random policy with 21 players with 9SL-2SR (blue) and no communication (green)

**Accord** The accord trend does not see any main difference between the current experiment and the baseline. The only dissimilarity is present at the start of the iteration in which the plot shows a small hill which quickly smooths out.

**Suicide** On the other hand the suicide rate shows a drastic decrease starting at iteration  $20M$  and reaching its lowest value at step  $50M$ . The magnitude of the drop, 4%, implies that half of the agents previously voting for themselves are now redirecting those votes to the wolves; with this information we can confidently say that the communication channel allows a better coordination between the agents .

**Days** It is trivial to see how the day trend does not only mimic the winning rate but shows additional information. Between iteration  $20M$  and  $40M$  the average day length shows an increase of 0.4% which is usual as the villagers start to understand the game and win , but, from step  $40M$  until the end, the trend marks a strong decrease, 1%.

As mentioned in the baseline section earlier, the  $21P$  setting has a total of 210 leaves where 126 sees the villager win and 84 do not; even though the probabilities are respectively 11.62% and 88.38%, heavily biased towards the werewolves, the villagers have many more possible outcomes; taking this into account, we understand how the villagers have found a closer leaf who's outcome is a win, thus reducing the number of days and increasing their chances to win.

This is a remarkable results since it implies not only a high coordination, but a quick navigation of the environment.

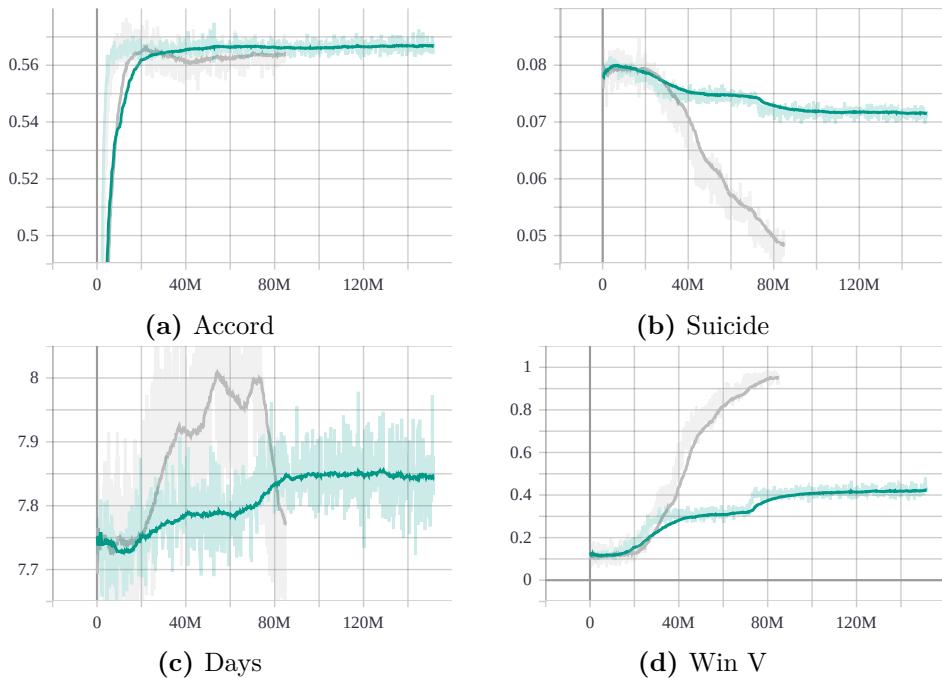
**Winning rate** Finally, the winning rate raises to the highest value encountered so far, 98%. This result proves both how the expansion of the communication channel is directly proportional to the villager's positive outcome and that there is seems to be a saturation point for  $SL$  after which the winning rate does not increase.

## 21SL-2SR

By maximizing the length of the communication channel while keeping the bit range, the reader can see in [Figure 3.20](#) how the final outcome is not positively influenced as it was for the  $9SL-2SR$  setting. The figure shows two multi-channel configurations together with the baseline value; in the following paragraphs these trend will be analyzed and compared to each other.

**Accord** Nothing is to be said for the agreement plot; every instance behaves in a similar fashion.

**Suicide** On the other hand, the suicide plot has very diverse trends depending on the setting. The  $9SL-2SR$  environment has the steepest decrease in the percentage of suicide, as discussed previously it reaches 4% in only  $40M$  steps. Differently, the  $21SL-2SR$  game, takes  $85M$  steps to approach 5%; this slow down can indicate that there is no linear behaviour when it comes to match outcome and signal shape, indeed it seems that various combination yield very dissimilar results.



**Figure 3.20.** Metric value for random policy with 21 players with 21SL-2SL (gray) and no communication (green)

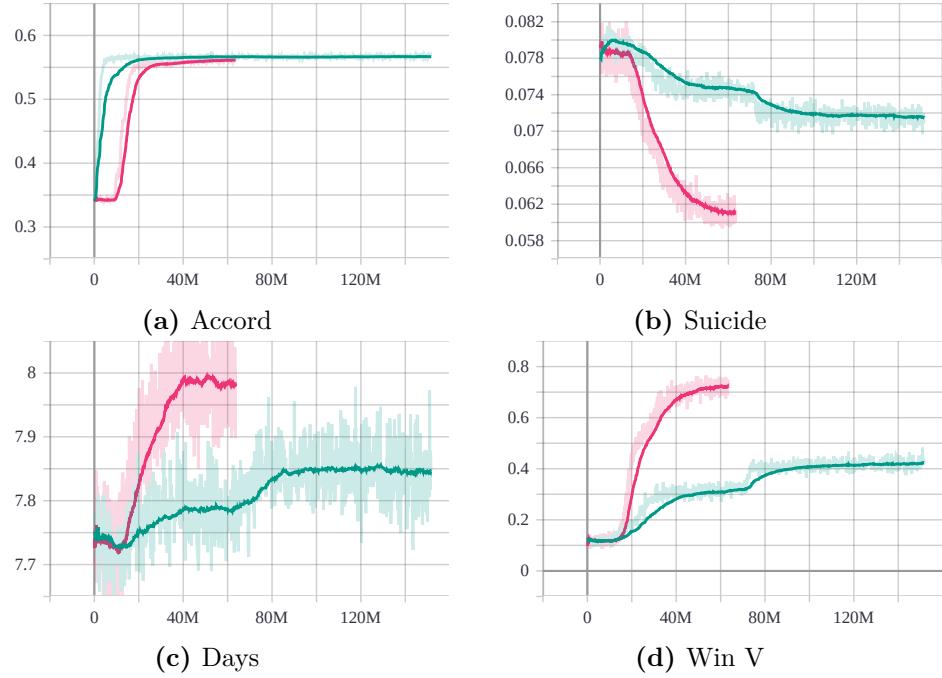
**Days** Another important factor we must consider is the average day length. As can be seen from the figure, the current setting yields a higher average day length than the baseline setting although the winning rate is much greater. This result can be again mapped to the choosing of a tree branch which has as a leaf a winning outcome but it involves a longer path too; by the start of the  $80M$  step this trend seems to decrease, but further training can confirm this hypothesis.

**Winning Rate** Finally, the fixed value is approximately the same for both the multi-channel configurations with a  $\pm 5\%$  difference. The major difference lies in the number of steps it takes to bring both the training to their maximum; for the previous setting a total of  $40M$  steps seems to be enough, while in the current one, a minimum of  $80M$  is required. Thus we can conclude that, for the bit ranged communication, more channels are not directly correlated with a better performance, indeed the opposite is true.

1SL-21SR

As can be seen from Figure 3.21, the introduction of an extended range greatly increases the agents' performance both in the outcomes and in speed.

**Accord** The accord metric shows how the introduction of the additional channel range challenges the agreements between the players; in fact the value stays constant for  $10M$  more steps before abruptly increasing to the maximum value. It is worth noticing that in this case there is not much more accordance between the player



**Figure 3.21.** Metric value for random policy with 21 players with 1SL-21SL (magenta) and no communication (green)

as there is in the 9P setting, indeed the only difference is the speed at which the players get to the fixed value.

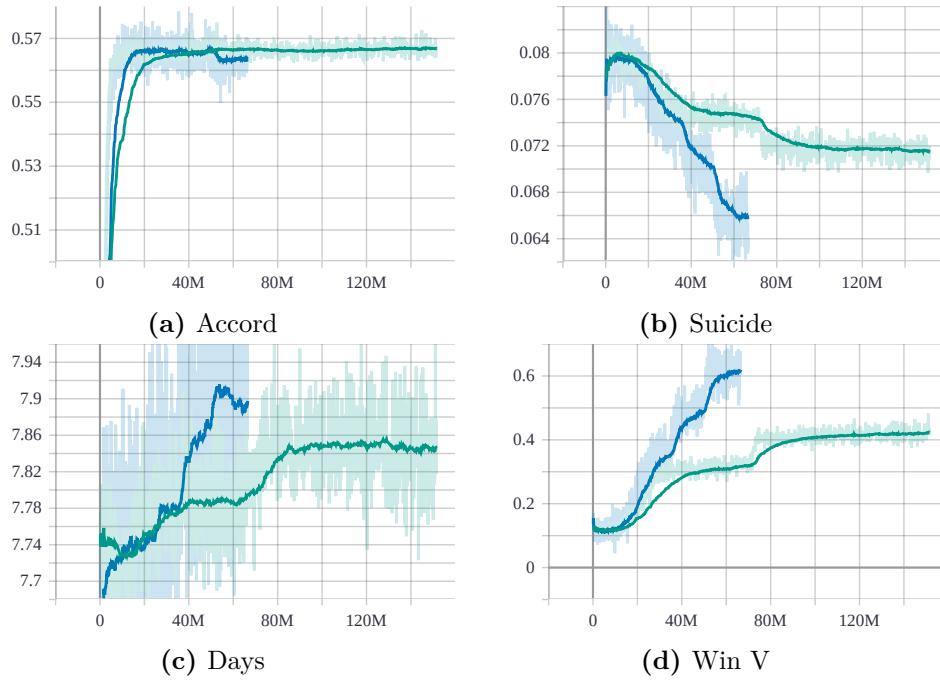
**Suicide** On the other side the suicide trend diverges significantly from the previous setting. As soon as around the  $20M$  step, this metric drops quickly by 1.2% reaching a value of 6.2%; the latter, together with the next parts, shows how much of the winning is given by fewer players voting for themselves.

**Days** The average day's length anticipates what shown in the next plot, that is a higher performance for the villagers. It is worth noticing how the small dune present at the start of the training is still there for this experiment, indeed with a steeper descent at the start of the game.

**Winning rate** Finally, the winning rate sees a sharp increase during the first  $30M$  steps, after which the value stabilizes at 74%. Referring to [Table 3.2](#), it is easy to see how the probabilities are now overturned; the introduction of a single integer is enough for the villager to dominate the game.

## 21SL-21SR

Before the comparison, we present the results of a setting in which the communication is maximized, that is both the signal length  $SL$  and the signal range  $SR$  are greatest.



**Figure 3.22.** Metric value for random policy with 21 players with 21SL-21SL (blue) and no communication (green)

**Accord** The agreement value sees a slight delay in reaching the convergence limit; moreover the limit is achieved with a sharp increase very similar to a sigmoid function.

**Suicide** As shown in the previous results, the suicide trend decreases sharply compared to the no communication setting.

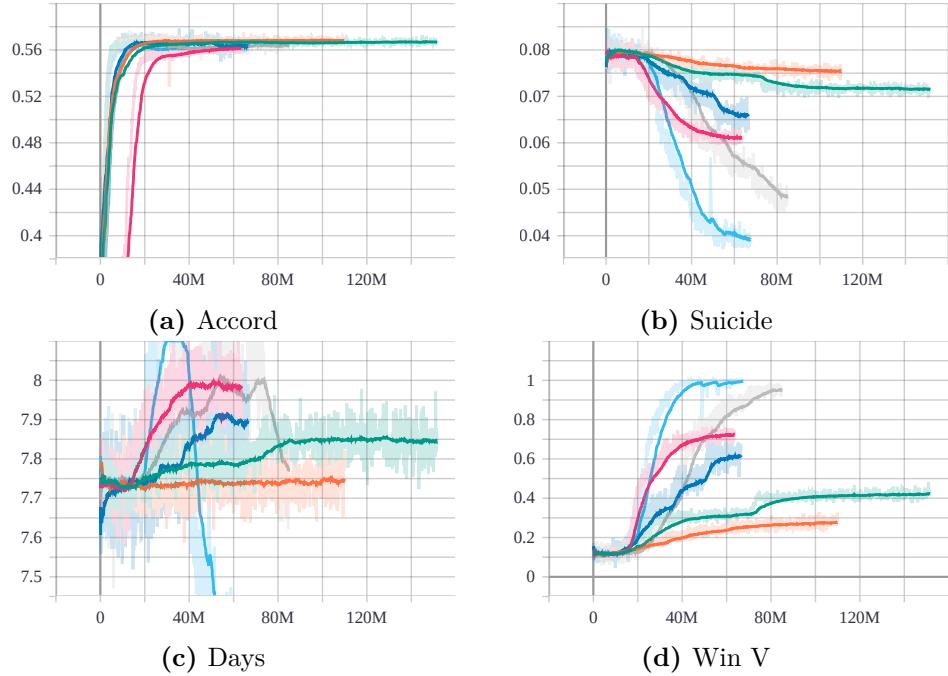
**Days** No major information can be extracted from this plot, indeed the average day length increases during  $40M$  iterations. Upon reaching 8, the trend saturates and stops.

**Winning rate** Finally, the winning trend is similar to the suicide one. In just  $40M$  iterations, from step  $20M$  to step  $60M$ , it reaches its limit at approximately 60%. We must say that the training time was increasing too much for all these experiments so we didn't train this last one as much as the others due to a lack of time; future research should investigate better the outcome of this setting.

### Comparison

In conclusion, the comparison between all the above mentioned settings is showed in [Figure 3.23](#).

**Accord** There are no significant differences between these trends. They all reach a value of  $56\% \pm 1\%$  in a fairly short amount of time, the only one showing a delayed



**Figure 3.23.** Metric value for random policy with 21 players with: 21SL-21SL (dark blue), 1SL-21SR (magenta), 21SL-2SR (gray), 9SL-2SR (light blue), bit communication (orange) and no communication (green)

behavior is  $1SL - 21SR$  setting which converges to the maximum level in  $15M$  more steps.

**Suicide** This metric shows much more variance than the previous one, indeed there is no common stability value for most of the setting. It is trivial to see that the ones showing more similarity are the no communication (orange) setting and the boolean one (magenta); given enough steps they would probably converge to the same value.

The  $1SL - 21SR$  environment (red) is the first one to decrease but shows a limited trend; in fact its behavior is similar to the one of a squeezed sigmoid function where most of the other plots have a linear trend.

The remaining settings,  $21SL - 21SR$  (green),  $21SL - 2SR$  (light blue) and  $9SL - 2SR$  (dark blue), shares a common linear trend. They all decrease rapidly with different slopes; the  $9SL - 2SR$  slows its decreasing approaching step  $75M$ .

**Days** Here we notice three different kind of behaviours.

The first one regards the setting with no and boolean communication; the number of days does not change much during the execution and stays approximately the same. Then there are the  $1SL - 21SR$  and  $21SL - 21SR$  environments where the average day length increases proportionally with the winning rate; in such cases the villager learn to redirect the outcomes to a winning branch farther always from the starting point compared to the werewolves winning.

Finally there are the situations in which the number of days shows an increase, as

the previous ones, and then decreases. Such behavior are found in the  $21SL - 2SR$  (light blue) and  $9SL - 2SR$  (dark blue) settings, where the villagers find a closer winning branch.

**Winning rate** In conclusion, there are two types of trends for the winning rate. To the first one belongs the  $21SL - 21SR$  (green), boolean and no communication settings; in those the rate shows one or more steps in which the rate increases rather than a constant linear development. This kind of behavior can be directly mapped to a difficulty in choosing the right strategies, thus changing them abruptly. Then, for the other trends, the increase is exponential at the start until it reaches the turning point where it becomes logarithmic.



# Chapter 4

## Discussion

In this thesis our aim was to find out if the introduction of a communication channel could increase the winning ratio of one of the two groups of players, the villagers. Moreover we investigated how the shape of such channel influences the outcome of a match and its performance by varying both the length and the range of the channel. Finally we let both parties learn alternatively to see how the cooperation and competitiveness induce the agents to change the strategies used so far, inspecting the field of non stationary learning.

In the following section we will answer specifically to all these questions using the literature as support.

### 4.1 Nine Players

In the result [Section 3.2](#) we considered a total of 9 cases, spanning three policies across three different communication configurations.

**Baselines** First we showed how an environment with no communication channel behaves closely to what mentioned in the baselines [Section 3.1](#) except when it came to the revenge policy ([Subsubsection 3.2.1](#)) which produced significant learning despite the absence of the channel itself; we came to the conclusion that this particular policy is easy to spot because the wolves are not protected by the randomness of their targets nor by the majority of the union policy.

**Bit communication** Then we saw how the introduction of a single bit for communication purposes ([Subsection 3.2.2](#)) exponentially increased the villager winning rate in all the policies treated so far. This gain was also exhibited in the performance since the agents took on average half the iteration necessary to reach an optimal learning point.

With this result only we proved our first hypothesis, that is the introduction of a communication channel indeed helps the agents coordinate much better and speeds the training up.

**Multi channel communication** Next we questioned how different communication settings could influence the outcomes of the game, so the multi-channel communication ([Subsection 3.2.3](#)) experiment was performed.

In the latter we investigated three different kind of channels by either maximizing the signal range ( $SR$ ), the signal length ( $SL$ ) or both.

In the first case([1SL-9SR](#)), where  $SR$  was maximized, the results were similar to the bit communication setting with a delayed factor; which brought us to the conclusion that more space for information does not directly maps with a better overall performance.

The second experiment ([9SL-2SR](#)) yielded the best results in which the agents were able to maximize their winning rate three times faster than in the previous cases. During the latter, we encountered an odd error which we studied and discarded as a machine performance drop.

Finally, the last observation ([9SL-9SR](#)) where we maximized both the range and the length of the signal, didn't show much improvement compared to the bit communication setting. Indeed the only upside was the speed of learning which was slightly optimized; on the other hand the winning rate tended to be the same.

**Findings** We found that the introduction of different forms of communication greatly increases the performance of the agents. In particular we observed that a boolean signal range is preferred to an integer one; the motivation behind this may lay in the duality of the roles or, more easily, the nature of the machine itself which communicates with bits to begin with.

Moreover we found how much of the villagers' winning rate is given by the agents not voting for themselves while keeping the accord level to a maximum; this implies better coordination which is possible only when there is a sufficient amount of communication present in the environment.

Finally we notice that there is no linear map between the amount of communication permitted, i.e.  $SL$  and  $SR$ , and the overall performance. Indeed there seems to be an optimal combination of the two which depend mainly on the signal length.

## 4.2 Twenty-one Players

Later on ([Section 3.3](#)), we studied the setting in which 21 players, 4 werewolves and 17 villagers, were able to play. In this experiment we limited our research to a single policy, the random one, for computational reasons.

**Baselines** First we analyzed the winning probabilities with the expanded tree ([Table 3.2](#)). Here we found out that the villagers were more likely to win compared to the nine player setting ([Table 3.2](#)) and there are many more possible outcomes where the villagers find a winning leaf.

Then we trained the agents in a non communicative settings ([Subsection 3.3.1](#)) and found that the winning rate could increase way above the theoretical value; this comes as no surprise since we saw how the villagers are not forced to spot a werewolf in the first day to win. Moreover, we noticed how there is no significant divergence between the accord and suicide trends between the 21P and 9P settings <sup>1</sup>; with this we concluded that communication is not as essential in the 21P setting as it is in the 9P.

**Bit Communication** Then we studied the environment in which a single bit could be used as a mean of communication ([Subsection 3.3.2](#)) where we compared the results with the previous case. Here we noticed how the final outcomes degraded instead of increasing as in the 9P setting; indeed the final winning rate was 20% lower than the no communication environment.

**Multi-channel communication** Next, the variation of both the signal length and the signal range was analyzed ([Subsection 3.3.3](#)).

In the first part we increase  $SL$  to 9 ([9SL-2SR](#)) and register a much greater performance in both time and value. With this setting the agents were able to correctly spot the werewolves almost 100% of the times, contemporaneously decreasing the average day length to a minimum ever registered.

Then we investigate the changes when the communication channel was allowed to use all the available space ([21SL-2SR](#)) and find out that the overall performance did not change much with respect to the previous case. Indeed the final outcome was slightly worse, 5%, and delayed by a factor of circa  $\times 2$ .

Furthermore, we switched parameter and increment the signal range while keeping the length to a minimum ([1SL-21SR](#)). We found a significant increase in the performance for what regards the baseline and bit communication values, which leads us to the conclusion that this kind of setting in which more players are involved favor the diversity of the communication channel rather than the length.

Finally , we maximized both  $SL$  and  $SR$  ([21SL-21SR](#)) and show how the performance increased although it did not surpass the value reached in the [9SL-2SR](#) setting.

**Findings** Differently from the 9P setting, here we found how the expanded tree has many more possible outcomes for the agents. This results implies that the outcomes reported above may vary greatly between runs even when they have the exact same parameters, thus further studies should average multiple runs for each setting in order to get accurate results.

Another approach to this problem could be the introduction of a curiosity based parameter in the algorithm [[Pathak et al., 2017](#)], but we leave this to future research.

Although we do not have certain results for each outcome, we see that most of the communication settings yield better outcomes than the one in the no communication baseline environment; thus we can confidently say that the addition of a communication setting yields better performances.

---

<sup>1</sup>Once the values were appropriately scaled by the proportion of roles.

On the same line as the previous settings, we see that the dimension of the communication channel, in both range and length, is not directly mapped to a better overall performance; indeed the best results are yield by the *9SL-2SR* environment where the range is minimum and the length is arbitrary. Again this show that there is some kind of optimal combination which could potentially lead to the best results in the shorted amount of time if investigated sufficiently, but, due to a lack of computational resources and time, we are able to analyze a small set of possible combinations.

Further research in this direction could try to have the signal range and/or length as a controllable parameter of a centralized training system, where the algorithm chooses the best combination of parameter based on a time-memory and performance constraint.

# Bibliography

- [Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
- [Ahmed et al., 2018] Ahmed, Z., Roux, N. L., Norouzi, M., and Schuurmans, D. (2018). Understanding the impact of entropy on policy optimization. *arXiv preprint arXiv:1811.11214*.
- [Amari, 1993] Amari, S.-i. (1993). Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196.
- [Baker et al., 2019] Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. (2019). Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*.
- [Barton et al., 2018] Barton, S. L., Waytowich, N. R., Zaroukian, E., and Asher, D. E. (2018). Measuring collaborative emergent behavior in multi-agent reinforcement learning. In *International Conference on Human Systems Engineering and Design: Future Trends and Applications*, pages 422–427. Springer.
- [Braverman et al., 2008] Braverman, M., Etesami, O., Mossel, E., et al. (2008). Mafia: A theoretical study of players and coalitions in a partial information environment. *The Annals of Applied Probability*, 18(3):825–846.
- [Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.
- [Buşoniu et al., 2010] Buşoniu, L., Babuška, R., and De Schutter, B. (2010). Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1*, pages 183–221. Springer.
- [Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

- [Chollet et al., 2015] Chollet, F. et al. (2015). Keras. <https://keras.io>.
- [Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.
- [Espeholt et al., 2018] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*.
- [Gilks, 2005] Gilks, W. R. (2005). M arkov chain m onte c arlo. *Encyclopedia of biostatistics*, 4.
- [Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- [Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.
- [Karlik and Olgac, 2011] Karlik, B. and Olgac, A. V. (2011). Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122.
- [Kempf et al., 2006] Kempf, G., Knudsen, F., Mumford, D., and Saint-Donat, B. (2006). *Toroidal embeddings 1*, volume 339. Springer.
- [Klopff, 1972] Klopff, A. H. (1972). *Brain function and adaptive systems: a heterostatic theory*. Number 133. Air Force Cambridge Research Laboratories, Air Force Systems Command, United ....
- [Leibo et al., 2019] Leibo, J. Z., Hughes, E., Lanctot, M., and Graepel, T. (2019). Autocurricula and the emergence of innovation from social interaction: A manifesto for multi-agent intelligence research. *arXiv preprint arXiv:1903.00742*.
- [Liang et al., 2017] Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Gonzalez, J., Goldberg, K., and Stoica, I. (2017). Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381*.
- [Luo et al., 2018] Luo, J., Green, S., Feghali, P., Legrady, G., and Koç, C. K. (2018). Visual diagnostics for deep reinforcement learning policy development. *arXiv preprint arXiv:1809.06781*.
- [McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.
- [Migdał, 2010] Migdał, P. (2010). A mathematical model of the mafia game. *arXiv preprint arXiv:1009.1031*.

- [Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- [Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.
- [Moritz et al., 2018] Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., et al. (2018). Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 561–577.
- [Nakamura et al., 2016] Nakamura, N., Inaba, M., Takahashi, K., Toriumi, F., Osawa, H., Katagami, D., and Shinoda, K. (2016). Constructing a human-like agent for the werewolf game using a psychological model based multiple perspectives. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8.
- [Pathak et al., 2017] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17.
- [Raileanu et al., 2018] Raileanu, R., Denton, E., Szlam, A., and Fergus, R. (2018). Modeling others using oneself in multi-agent reinforcement learning. *arXiv preprint arXiv:1802.09640*.
- [Samuel, 1959] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.
- [Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.
- [Schulman et al., 2015a] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897.
- [Schulman et al., 2015b] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.
- [Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- [Schuster and Paliwal, 1997] Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

- [Smallwood and Sondik, 1973] Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable markov processes over a finite horizon. *Operations research*, 21(5):1071–1088.
- [Sycara, 1998] Sycara, K. P. (1998). Multiagent systems. *AI magazine*, 19(2):79–79.
- [Tammina, ] Tammina, S. Transfer learning using vgg-16 with deep convolutional neural network for classifying images.
- [Tan, 1993] Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337.
- [Van Erven and Harremos, 2014] Van Erven, T. and Harremos, P. (2014). Rényi divergence and kullback-leibler divergence. *IEEE Transactions on Information Theory*, 60(7):3797–3820.
- [Wang and Kaneko, 2018] Wang, T. and Kaneko, T. (2018). Application of deep reinforcement learning in werewolf game agents. In *2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 28–33.
- [Wang et al., 2016] Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2016). Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*.
- [Watkins and Dayan, 1992] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.