Diseño y desarrollo de un agente negociador para la competición de Werewolf de la ANAC2019

Ramon Ruiz-Dolz and Javier Iranzo-Sánchez

Institut Valencià d'Investigació en Intel·ligència Artificial
Universitat Politècnica de València
Camino de Vera s/n. 46022 Valencia (Spain)
raruidol@dsic.upv.es, jairsan@upv.es

16 de abril de 2019

Resumen

El campo de la Inteligencia Artificial ha prestado gran atención al desarrollo de agentes inteligentes que sean capaces de jugar a diversos tipos de juegos. El desarrollo de estos agentes adquiere una complejidad adicional cuando se trata de el tema desde la perspectiva de sistemas multiagentes. Esto requiere el tratamiento de nuevos problemas como la información incompleta. En este trabajo, presentamos el proceso completo de desarrollo de un agente inteligente que incorpora el uso de técnicas de aprendizaje automático y argumentación para afrontar el problema de la información incompleta, demostrando experimentalmente que nuestra aproximación es capaz de extraer información útil sobre el comportamiento de otros agentes. Así mismo, se describe con detalle el dominio utilizado y el marco de la competición ANAC2019.

1. Introducción

En el marco del trabajo de la asignatura de HAIA hemos decidido realizar el desarrollo de un modelo de predicción de información oculta en juegos con información incompleta así como la implementación de un agente argumentador para la competición de agentes negociadores ANAC2019[1]. En este trabajo se realiza tanto el desarrollo de un modelo de predicción como su evaluación a partir de datos publicados de anteriores competiciones. Además también se exponen las distintas heurísticas de negociación así como el sistema argumentativo empleado para el desarrollo del agente implementado para la plataforma AIWolf[2]. AIWolf es una plataforma multiagente que permite jugar al juego del *Werewolf* mediante agentes autónomos inteligentes. Este dominio es realmente interesante desde el punto de vista de la investigación, puesto que varios campos como la argumentación, los sistemas multiagentes o la negociación confluyen en él. Al ser un juego de información incompleta eleva la dificultad a la hora de trabajar con el dominio, es por ello que consideramos interesante contar con modelos de predicción capaces de estimar con suficiente precisión la información oculta.

La estructura del trabajo es la siguiente. En la Sección 2 se detalla el dominio sobre el que se ha realizado este trabajo, se explican las reglas y se definen los roles existentes. En la Sección 3 se presenta la plataforma sobre la que se van a lanzar los agentes. Se presenta su arquitectura y los protocolos de comunicación entre agentes de los que dispone. En la Sección 4 se explica el proceso de modelado de oponentes realizado mediante tecnicas de aprendizaje automático. En la Sección 5 se introducen los conceptos fundamentales de la argumentación en sistemas inteligentes y se presenta el sistema argumentativo desarrollado en este trabajo. En la Sección 6 se explica la heurística de negociación implementada en el agente jugador. En la Sección 7 se introducen las principales normas de la competición de Werewolf en la ANAC2019 y finalmente, en la Sección 9 se presentan las conclusiones alcanzadas a lo largo de la realización de este trabajo.

[1] `http://web.tuat.ac.jp/~katfuji/ANAC2019/#werewolf`
[2] `http://aiwolf.org/en/`

2. Dominio de desarrollo

2.1. Reglas del juego

Werewolf es un juego que pertenece al arquetipo de los juegos de roles ocultos. Estos juegos tienen en común el hecho de que se les asigna un rol a cada jugador (normalmente de manera aleatoria) al comienzo de la partida, y este rol condiciona las habilidades, funcionamiento y objetivos del jugador durante la partida. Adicionalmente, los jugadores no tienen información perfecta sobre los roles del resto de jugadores, por lo que una parte importante de una estrategia ganadora consisten en averiguar información sobre el resto de jugadores a la vez que se mantienen ocultos nuestros objetivos del resto de jugadores.

En Werewolf existen dos grandes equipos: una mayoría desinformada, los aldeanos (*villagers*), que desconocen por completo el rol del resto de jugadores, y una minoría informada, los hombres-lobo (*werewolves*), que conocen la identidad del resto de hombres-lobo. El objetivo del cada equipo es acabar con el equipo contrario.

2.2. Desarrollo de la partida

Una partida de Werewolf consiste en una serie de turnos de Día y Noche que se van alternando uno detrás de otro. Durante el Día, se realiza una votación para decidir que persona será ejecutada ese día, en un intento de los aldeanos de combatir el problema de los ataques de hombres lobo. Durante este proceso, los jugadores pueden comunicarse entre ellos. Así pues, el objetivo de los aldeanos es coordinarse para poder ejecutar a un lobo durante el día, ya que es la única manera que tienen de acabar con los lobos, mientras que los lobos tienen que intentar evitar ser descubiertos para que los aldeanos se vayan matando entre ellos, causando desconfianza entre los aldeanos. Al finalizar este proceso de deliberación, aquel jugador que haya recibido un mayor número de votos es ejecutado y eliminado de la partida.

El turno de Noche sigue al turno de Día. Durante la noche, los lobos salen de caza y atacan la aldea. Los lobos se ponen de acuerdo entre ellos sobre que jugador atacar, y el aldeano seleccionado muere y es eliminado del juego. El proceso de ataque de los lobos sucede todos los turnos, y es su mejor herramienta para ir eliminando a los aldeanos de manera garantizada.

Adicionalmente, durante la fase de Noche, otros roles especiales también actúan. Evidentemente, esto solo sucede cuando alguno de esos roles están presentes en la partida.

Es importante destacar que no se revela el rol de los jugadores muertos, lo que añade una mayor incertidumbre a la partida. [3]

El esquema de desarrollo de una partida se muestra de manera gráfica en la Figura 1.

2.3. Roles especiales

En la versión de Werewolf a tratar, existen 4 roles especiales a parte de los básicos Villager y Werewolf, que son los siguientes:

- **Seer**: Durante la noche, el Seer puede escoger a otro jugador para averiguar más información sobre él. El Seer recibe únicamente el equipo al que pertenece el jugador seleccionado (es decir, aldeano u hombre lobo), pero no recibe información sobre su rol. Este es el único rol que permite a los aldeanos obtener información fidedigna sobre la identidad de los jugadores, y representa una de las mayores bazas de los aldeanos.

- **Medium**: El Medium es informado sobre si la persona que fue ejecutada durante el día era un hombre lobo o no. Este rol permite tener una idea de como está evolucionando la partida, e incluso puede servir para comprobar si existen patrones en las votaciones que nos hagan sospechar de algún aldeano en particular.

- **Bodyguard**: El Bodyguard tiene la capacidad de proteger a otro jugador del ataque de los lobos. Durante la noche, el Bodyguard selecciona a un jugador, que recibe inmunidad a los ataques de los lobos durante esa noche. No hay restricción sobre la selección, y el Bodyguard puede incluso seleccionarse a si mismo. El Bodyguard tiene que intentar ir un paso por delante de los lobos y proteger a sus objetivos a los aldeanos importantes.

- **Possessed**: El jugador Possessed no tiene habilidades especiales, pero su condición de victoria es la contraria que un aldeano normal: El jugador Possessed gana la partida si ganan los lobos, por lo que tendrá que conspirar contra el resto de aldeanos si quiere salir victorioso. La identidad de este jugador no es revelado a los lobos al principio de la partida.

La presencia de estos roles especiales añade complejidad al juego y permite desarrollar una mayor serie de estrategias.

[3] Existen variantes del juego en las que si que se revela esta información.

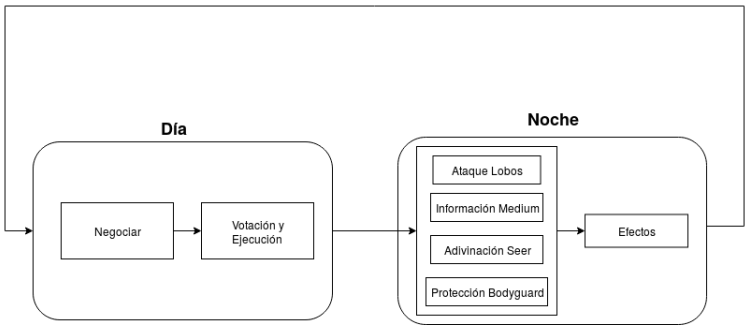

Figura 1: Esquema del desarrollo de una partida de Werewolf

3. Plataforma AIWolf

La plataforma multiagente AIWolf es una herramienta desarrollada por Fujio Toriumi, Hirotaka Osawa y Atom Sonoda entre otros investigadores, de las universidades de Tokyo y Tsukuba entre otras instituciones. El proyecto AIWolf[4] nace con el propósito de desarrollar agentes inteligentes capaces de coordinarse, cooperar y competir en un entorno con información incompleta. A diferencia de la mayoría de juegos con los que se han conseguido grandes hitos en el mundo de la inteligencia artificial, los juegos con información incompleta son una área todavía a explorar. El juego del Werewolf es realmente interesante desde el punto de vista de la inteligencia artificial puesto que obliga a los jugadores a interactuar entre sí, argumentar, persuadir y juzgar la información recibida. Siendo todos temas candentes en el mundo de la investigación actual.

En los siguientes apartados, siguiendo la información publicada en [9], se describen los principales elementos de AIWolf como son el servidor y los clientes. Además también se explica el funcionamiento de una ejecución del juego en esta plataforma.

3.1. Arquitectura de la plataforma

Debido a su naturaleza, la plataforma tiene una arquitectura cliente-servidor. Esta permite realizar la comunicación mediante el protocolo TCP/IP o bien mediante una API interna.

3.1.1. Servidor

En AIWolf, el servidor desempeña un papel principal en el desarrollo de las ejecuciones de las partidas. El servidor espera activo a que se conecten todos los clientes necesarios (agentes) para iniciar la partida, y en ese momento comienza el juego. Todo lo correspondiente con el desarrollo de la partida, las normas del juego y la gestión de los agentes es llevado a cabo por el servidor de AIWolf. Concretamente el servidor pide a los clientes que comuniquen sus acciones y estos responden cuando se les requiere siguiendo un diagrama de flujo similar al de la Figura 2.

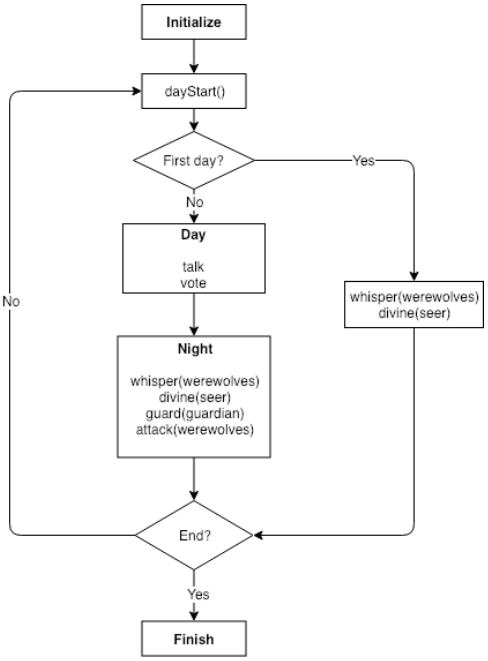

Figura 2: Diagrama de flujo de una partida en AIWolf.

3.1.2. Cliente

Cada cliente en AIWolf consiste en un agente jugador. En una ejecución de la plataforma AIWolf cada cliente puede ser diferente, de igual forma que en una partida del juego clásico (Werewolf), cada jugador humano puede tener una estrategia distinta o ser mas o menos crédulo en función de gran cantidad de factores. Es por ello, que el verdadero

[4] http://aiwolf.org/en/

reto consiste en ser capaz de encontrar una estrategia para el agente jugador, que le permita dominar a los demás agentes. Además, cada cliente debe ser capaz de jugar con cualquiera de los roles existentes. Tal y como muestra la Figura 3, la asignación de roles viene dada por el servidor una vez la partida cumple los requisitos para ser iniciada.

3.2. Comunicación entre agentes

La interacción es algo fundamental en un juego de estas características. Para comunicarse, los agentes utilizan un protocolo comunicativo propio de la plataforma AIWolf. Este protocolo se caracteriza por su funcionamiento por turnos. Cada agente puede escoger enviar hasta un mensaje por turno, además también puede optar por mantenerse al margen de la conversación un turno determinado. El límite de mensajes por cada agente en un dia determinado es de 10 mensajes máximos. Existen dos momentos en los que la comunicación entre agentes es requerida, la fase durante el día en la que todos los agentes comparten mensajes de manera pública y una fase de conversación nocturna donde únicamente los agentes con el rol Werewolf pueden comunicarse. En total existen 15 tipos distintos de primitivas mediante las cuales los agentes pueden transmitir sus pensamientos o sus decisiones.
Primitivas que expresan conocimiento:

- **estimate(Agent, Role)**: Mediante este protocolo, un agente determinado es capaz de expresar sus sospechas sobre el rol jugado por otro agente dentro de la partida.
- **comingout(Agent, Role)** Este protocolo permite a un agente revelar su rol. Esta información no tiene porque ser cierta.

Primitivas sobre intención de realizar acciones:

- **divination(Agent)** Mediante este protocolo, un agente comunica su intención de adivinar a un agente determinado.
- **guard(Agent)** Este es el protocolo mediante el cual se informa que se va a proteger a un agente determinado.
- **vote(Agent)** Protocolo para comunicar la elección del voto para la siguiente ronda de votación.
- **attack(Agent)** Protocolo mediante el cual se expresa el objetivo al cual se pretende atacar.

Primitivas sobre acciones realizadas previamente:

- **divined(Agent, Species)** Mediante este protocolo, un agente comunica el resultado obtenido tras realizar una adivinación.
- **identified(Agent, Species)** Similar al anterior, mediante este protocolo un agente informa del equipo al que pertenece el agente ejecutado.
- **guarded(Agent)** Este es el protocolo mediante el cual se informa que ha protegido a un agente determinado.
- **voted(Agent)** Protocolo para comunicar la elección del voto en la anterior ronda de votación.
- **attacked(Agent)** Protocolo mediante el cual se expresa el objetivo del último ataque realizado.

Primitivas para expresar acuerdo o desacuerdo:

- **agree(Talk_id)** Protocolo para mostrar acuerdo con un mensaje de un agente determinado.
- **disagree(Talk_id)** Justo al contrario que en el anterior caso, este es el protocolo mediante el cual se comunica el desacuerdo con un mensaje de un agente determinado.

Protocolos para controlar el flujo de la conversación:

- **skip()** Este protocolo permite a un agente saltarse su turno para hablar y esperar hasta el próximo turno.
- **over()** Similar al anterior, haciendo uso de este protocolo el agente salta el turno actual y acepta finalizar la conversación.

Todos estos protocolos permiten a cada agente expresar sus opiniones o engañar a los demás agentes. Además de la capacidad de lanzar cualquier tipo de mensaje siguiendo las primitivas descritas, en AIWolf también es posible trabajar con distintos tipos de operadores, concretamente dispone de 8 tipos diferentes de operadores disponibles.
Operadores para indicar peticiones e información:

- **request(Agent)** Operador mediante el cual se le solicita a un agente realizar una determinada acción.
- **inquire(Agent)** Operador mediante el cual se le solicita una información determinada a un agente concreto.

Operadores de razonamiento:

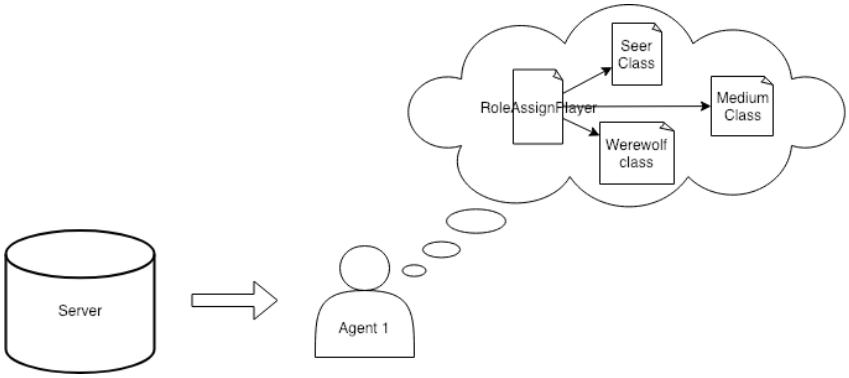

Figura 3: Asignación de roles a cada agente.

- **because(Frase1)(Frase2)** Operador mediante el cual se justifica la Frase2 a partir de la información existente en la Frase1.

Operadores de tiempo:

- **day(Num)(Frase)** Operador mediante el cual se puede contextualizar una frase en un dia determinado.

Operadores lógicos:

- **not(Frase)** Operador mediante el cual se niega una frase determinada.
- **and(Frase1)(Frase2)(...)** Operador mediante el cual se afirman múltiples frases a la vez.
- **or(Frase1)(Frase2)(...)** Operador mediante el cual se afirma una de todas las posibles frases.
- **xor(Frase1)(Frase2)** Operador mediante el cual se indica exclusión mutua entre la veracidad de ambas frases.

De todos estos operadores es importante destacar los operadores de razonamiento, puesto que mediante esta clase de operadores nos será posible construir un marco argumentativo. Esto es realmente importante puesto que en un juego donde la negociación y la argumentación son elementos cruciales para el funcionamiento, es muy importante ser capaces de valorar los distintos argumentos recibidos a lo largo de la fase de comunicación y tomar las decisiones más adecuadas.

4. Modelado de oponentes mediante aprendizaje automático

En este sección se describe el desarrollo de un sistema automático de detección de roles, de cara a mejorar el comportamiento de nuestro agente a la hora de enfrentarse a otros.

El campo del aprendizaje automático ha prestado atención particular al aprendizaje de comportamientos, de manera completamente automática, por parte de agentes. El algoritmo Deep-Q [6] ha sentado la base para un conjunto de investigaciones posteriores sobre aprendizaje por refuerzo utilizando técnicas de deep learning (deep reinforcement learning). Trabajos posteriores han extendido los resultados al caso multi-agente [4, 8], con las complicaciones que esto implica. La desventaja que tienen todos estos métodos basados en aprendizaje por refuerzo es que requieren de un número muy elevado de partidas (que normalmente ronda los millones) para aprender a actuar correctamente.

Si bien el caso ideal sería un agente autónomo que aprendiera a jugar sin intervención humana, los dos problemas mencionados (existencia de otros agentes y restricciones computacionales) hacen que, actualmente, no sea una aproximación posible.

En esta sección se presenta el desarrollo de un módulo de aprendizaje automático que, utilizando técnicas que se adecuan a las restricciones del problema que estamos tratando, sea capaz de estimar información oculta de los demás agentes, como por ejemplo su rol.

Formalmente, si en una partida tenemos j_1^J jugadores y r_1^R roles, y hemos observado una serie de eventos x_1^N, desearíamos conocer la probabilidad de que cada jugador j tenga asignado un rol m:

$$p(y_j = r_m | x_1^N) \ \forall j \in J \ \forall m \in R \tag{1}$$

Mediante la regla de la cadena, se puede descomponer la probabilidad de la siguiente manera:

$$p(y_1^J | x_1^N) = \prod_{j=1}^{J} p(y_j | x_1^N, y_1^{j-1}) \tag{2}$$

Una primera aproximación consiste en asumir independencia entre los roles de los jugadores.

$$p(y_1^J|x_1^N) := \prod_{j=1}^{J} p(y_j|x_1^N) \tag{3}$$

Resulta evidente que esta asunción no es cierta, ya que no todas las combinaciones de roles son válidas. A pesar de eso, es un punto inicial a partir del cual desarrollar un sistema de estimación de roles.

Adicionalmente, también hemos hecho la asunción de que el rol de un agente no depende de todos lo eventos de la partida, si no que depende únicamente de los eventos que ha realizado ese agente, $\tilde{x}_1^K$.

$$p(y_j|x_1^N) := p(y_j|\tilde{x}_1^K) \tag{4}$$

A partir de estos supuestos, podemos desarrollar un modelo θ_j que estima las probabilidades de los diferentes roles de un agente, en base a kas acciones que ha realizado durante una partida. Para entrenar este modelo, utilizaremos las acciones que ha realizado ese agente en partidas anteriores.

Vamos a extraer un único vector de características de cada partida, dónde cada elemento del vector indica el número de veces que el agente ha realizado una performativa durante la partida. A partir de este vector y el rol asignado a cada agente, se puede entrenar un clasificador de manera estándar. Nuestra aproximación al tratamiento de secuencias en base de un único vector está basado en una versión simplificada de [5], trabajo en el que se suma la representación de cada elemento de una secuencia para obtener un vector que represente a la secuencia entera. La propuesta que presentamos se basa en sumar únicamente el vector one-hot que indica que performativa se ha realizado en cada momento, pero se podría ampliar extrayendo una representación automática o *embedding* para cada performativa antes de sumarlas. El modelo utilizado es un modelo de Regresión Logística implementado en el paquete de software Scikit-learn [7].

Para realizar los experimentos de esta sección, vamos a utilizar los *logs* de la GAT2017, disponibles públicamente.[5] Este conjunto de datos contiene información sobre cerca de 1 millón de partidas, en las cuales se incluían, entre otros, 10 agentes inteligentes programados por diferentes equipos.

Un experimento necesario para validar nuestra aproximación consiste en medir el rendimiento del clasificador en función del número de partidas que se hayan utilizado para entrenarlo. Es de esperar que, a mayor cantidad de partidas, mejor sean los resultados del modelo. Además, resultara necesario que el clasificador tenga un buen funcionamiento aunque se haya entrenado con un número de partidas reducido, ya que nuestro objetivo es utilizarlo en una aplicación real, por lo que necesitamos obtener resultados bajo condiciones reales. En concreto, en el caso de la ANAC2019 (cuyas normas están detalladas en la Sección 7) se realizan una serie de rondas, donde cada una consiste en 100 partidas contra los mismos agentes oponentes, por lo que poder aprender información discriminativa con un número muy reducido de muestras es crucial para obtener buenos resultados.

Para este experimento, se ha entrenado un modelo para cada agente, utilizando un número variable de muestras (entre 10 y 1000), y se ha evaluado la capacidad de cada modelo de predecir los roles del agente con cuyas acciones fue entrenado, en un conjunto disjunto del de entrenamiento compuesto por 1000 nuevas partidas. El rendimiento del modelo se ha medido mediante la medida F1. Esta es una aproximación optimista, ya que los resultados se miden al final de la partida, por lo que nuestro clasificador tiene la máxima información disponible.

Los resultados de los experimentos, medidos mediante la medida F1, calculada a nivel micro, se muestran en la Figura 4. En la Figura 5, pero con la F1 calculada a nivel macro. Los resultados se comparan con una aproximación *Baseline* que consiste en asignar al agente un rol de manera aleatoria, en base a la probabilidad a priori que tiene cada rol.

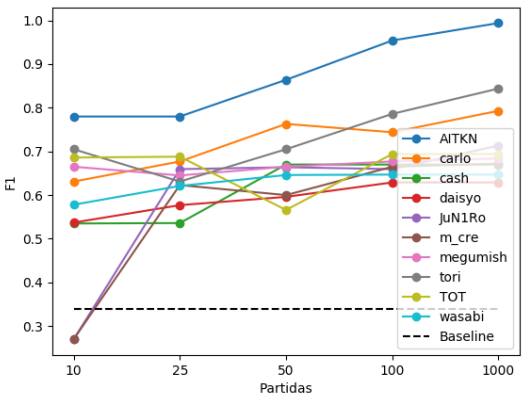

Figura 4: Medida F1 (micro) en función del número de partidas observadas.

A partir de estos resultados, se puede extraer que, a pesar del elevado número de asunciones que hemos realizado, nuestra aproximación es capaz de extraer patrones útiles que sirven para realizar bue-

[5] http://aiwolf.org/resource

nas predicciones sobre el rol asignado a cada agente. Los resultados también ponen de relevancia el problema que presenta trabajar con pocas muestras.

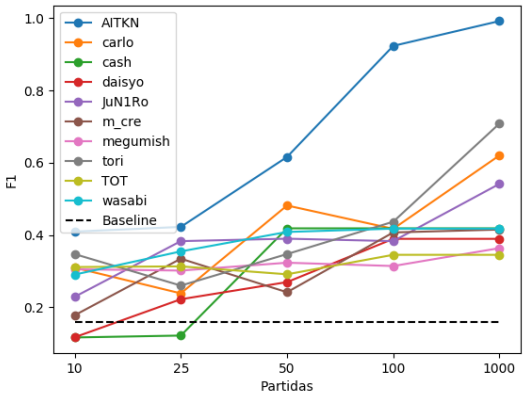

Figura 5: Medida F1 (macro) en función del número de partidas observadas.

Además de estudiar los efectos del número de muestras utilizadas para entrenar, también es necesario evaluar el comportamiento de nuestro modelo en función del momento de la partida. Los experimentos anteriores se han realizado utilizando una situación optimista, en la que el modelo realiza la predicción al final de la partida. De cara a utilizar el sistema en una situación real, es necesario analizar el comportamiento del modelo si analizamos la corrección sus predicciones en un punto intermedio de la partida, utilizando únicamente los eventos que se han producido hasta el momento, en lugar de al final de la partida, cuando están disponibles todos los eventos. Los modelos se han entrenado con las muestras de 50 partidas, de manera que simulan los resultados de una posible estrategia que consisten en entrenar un modelo durante las primeras 50 partidas de una ronda, y utilizarlo como ayuda en las 50 últimas. Los resultados de la medida F1, evaluada a nivel macro, se muestran en la Figura 6.

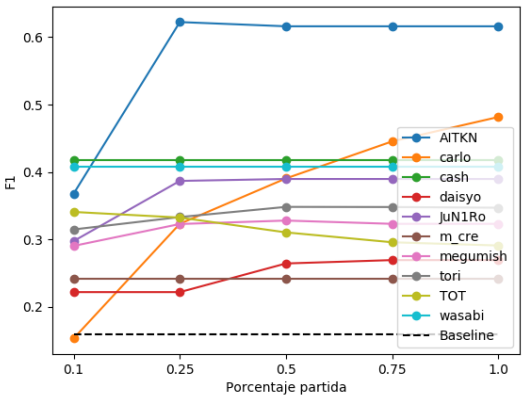

Figura 6: Rendimiento del modelo en función del tiempo transcurrido de partida

Los resultados demuestran la capacidad del modelo de realizar predicciones precisas en cualquier momento de la partida, lo que nos lleva a pensar que los agentes estudiados tienen comportamientos marcadamente diferentes dependiendo de su rol en los primeros turnos de la partida. Podemos observar como el rendimiento del modelo mejora a medida que avanza la partida en 6 de los agentes estudiados, mientras que el rendimiento disminuye en 2 de ellos. Creemos que este comportamiento se debe a que nuestra extracción de características, basada en cuentas, no mantiene la información secuencial sobre el orden de los eventos. Al carecer de esta información secuencial, se puede dar el caso que ciertos eventos del principio de la partida que pueden resultar muy discriminativos para estimar el rol de un agente, se van diluyendo a medida que avanza una partida y crece el número de eventos.

Por último, es interesante evaluar si nuestra tiene capacidad de generalización a la hora de aplicarse a nuevos agentes para los que no se haya nunca, y que no dispongamos la capacidad de re-entrenar nuestro modelo, debido a, por ejemplo, restricciones temporales o condiciones experimentales. Nuestra hipótesis es que, si este nuevo agente actúa de manera similar a los agentes utilizados para entrenar el modelo, obtendremos buenos resultados. En cambio, si este nuevo agente tiene un comportamiento muy particular, o incluso actúa de la manera contraria a los agente vistos previamente, obtendremos malos resultados. Los resultados de este experimento, dependerán por tanto mucho del conjunto de agentes utilizados, y de la mentalidad con la que afrontan este problema los creadores de estos agentes inteligente. Siguiendo este aproximación, nuestro siguiente experimento consiste en entrenar un clasificador para el agente j utilizando partidas del

conjunto de agentes $\{i \in J | i \neq j\}$, y comprobar su comportamiento a la hora de estimar el rol en un conjunto de 1000 partidas del jugador j, el mismo conjunto que en los casos anteriores. Los resultados de la medida F1, evaluada a nivel macro, se muestran en la Figura 7.

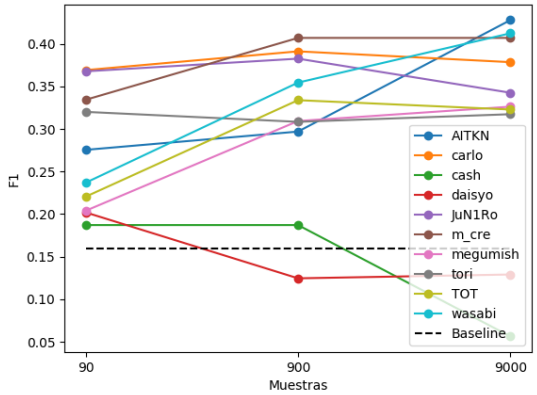

Figura 7: Rendimiento del modelo en función del tiempo transcurrido de partida

De los resultados se puede extraer que, para este conjunto de agentes, si que es posible una cierta generalización del conocimiento aprendido a partir de partidas de otros agentes, aunque con peores resultados que cuando podemos estimar nuestro modelo tras observar el comportamiento del agente a predecir. Hay dos agentes para los cuales los resultados son peores que la Baseline aleatoria, por lo que cabe concluir que su comportamiento es muy diferentes del resto de agentes, y por esta razón resulta tan complicado predecir su comportamiento sin haber utilizado muestras de ese agente durante el entrenamiento.

5. Marco argumentativo para la mejora de la persuasión

Tomando el marco argumentativo abstracto de Dung [3] como punto de partida, se define un marco de argumentación abstracta como una tupla de dos elementos $<A, R>$ sea A el conjunto de argumentos y R el conjunto de relaciones entre los distintos argumentos. En nuestro caso vemos muy interesante la ampliación de estos marcos argumentativos propuesta en [1, 2] donde se proponen los marcos argumentativos basados en valores. Un marco argumentativo basado en valores (Value-Based Argumentation Framework, VAF) se define como,

Definition 5.1 (Value-Based Argumentation Framework)**.** Un VAF consiste en una tupla $<A,R,V,val,P>$ donde A consiste en el conjunto de argumentos, R consiste en el conjunto de relaciones sobre A, V es el conjunto de posibles valores asociables a los argumentos, val es una función que relaciona los elementos de A con los elementos de V y finalmente, P es el conjunto de posibles audiencias.

En este trabajo vamos a partir de las ideas principales que plantean los VAFs para adaptarlas a nuestro dominio según nuestras necesidades. El marco argumentativo diseñado para la implementación de este trabajo podría verse como una concretización del marco argumentativo basado en valores.

Definition 5.2 (Trust-Based Argumentation Framework)**.** Nuestro TAF consiste en una tupla $<A,R,\delta,P>$ donde A consiste en el conjunto de argumentos, R consiste en el conjunto de relaciones sobre A, δ es un factor de puntuación basado en la confianza para cada argumento dado un agente determinado y P es el conjunto de agentes involucrados en la argumentación.

A continuación se detalla el proceso de instanciación de este marco argumentativo propuesto dentro del dominio del juego.

5.1. Instanciación del marco argumentativo en el agente AIWolf

A partir de la definición formal del marco argumentativo propuesto, es necesario realizar una instanciación de este para nuestro dominio de uso concreto, el juego del Werewolf para la plataforma AIWolf. Tras realizar la implementación, cada elemento del marco tiene la siguiente forma:

- **Argumentos (A):**

  Tal y como ya se ha descrito en la Sección 3, la comunicación entre agentes sigue un protocolo predefinido. Es por esto que cada argumento existente en nuestro marco argumentativo estará compuesto por dos elementos, $\forall \alpha \in A$, $\alpha = (claim, support)$ donde $claim$ es la conclusión del argumento y $support$ es su soporte. Siguiendo la estructura del protocolo de AIWolf, un argumento *because(F1)(F2)* generaría un argumento $\alpha = (F2, F1)$.

- **Relaciones (R):**

  Puesto que el objetivo de la argumentación es escoger un único objetivo al que votar para su ejecución durante la fase del día, el resultado

de la argumentación debe ser votar a un agente en concreto. Una relación en nuestro marco se define como, $\exists r(\alpha_i, \alpha_j) \iff claim_i \neq claim_j$. En otras palabras, existirán relaciones de ataque entre todos los argumentos cuyo *claim* sea diferente.

- **Puntuación basada en la confianza (δ):**

  Para nuestro caso concreto, la puntuación $\delta(\alpha, p)$ para un argumento emitido por el agente $p \in P$ tendrá un valor determinado en función del número de mentiras realizado por ese agente. Formalmente definimos esta función como,

$$\delta(\alpha, p) = \frac{1}{n_p + 1} \tag{5}$$

  donde n_p es el número de mentiras realizadas por el agente p. Por lo tanto, el valor de nuestra función de puntuación consiste en una normalización del número de mentiras realizadas por cada agente participante en la conversación.

- **Agentes (P):**

  Finalmente el parámetro P del marco argumentativo consiste en una lista de todos los agentes que hayan participado en la conversación de un día determinado aportando como mínimo un argumento al diálogo.

6. Heurística de negociación: una aproximación basada en la detección de mentiras

Además del marco argumentativo definido con el objetivo de poder gestionar de forma adecuada la argumentación sucedida en el transcurso de la partida, también es importante hacer énfasis en la heurística de negociación desarrollada. Una heurística de negociación consiste en una estrategia a seguir durante el proceso de negociación entre agentes. En nuestro caso hemos decidido realizar una aproximación basada en el análisis y la detección de mentiras.

Tal y como se ha descrito anteriormente, Werewolf es un juego en el que la comunicación es primordial. A parte de las habilidades específicas de cada rol, la fase de comunicación es decisiva en el trascurso de la partida puesto que el número de jugadores se decrementa en uno. Por lo tanto, dependiendo del equipo al que pertenezca nuestro agente jugador, deberá seguir una gestión de mentiras diferente. El método que se sigue con tal de detectar una mentira consiste básicamente en llevar un control de incoherencias en los mensajes. Estas incoherencias pueden ocurrir, bien entre los mensajes enviados en la fase de negociación o entre la información privada de nuestro agente y la información publicada por otro agente.

Durante el transcurso de la negociación, en el caso de pertenecer al equipo de los aldeanos (Villager), se llevará a cabo un análisis y recuento de mentiras *reales*. Mientras que para el caso de los miembros del equipo de los lobos (Werewolves), además de las mentiras *reales* también se tendrán en cuenta como mentiras, aquellos comentarios que puedan levantar sospechas sobre su pertenencia al equipo de los hombres-lobo. Por lo tanto, también trabajará con mentiras *falsas*.

Como respuesta a las mentiras detectadas, nuestro agente jugador lanzará como *oferta*, un argumento razonando con la información incoherente como *support* y un mensaje de sospecha sobre pertenencia al equipo *Werewolf* como *claim*, tal y como se ha definido en la sección anterior. Siguiendo esta estrategia se tratará de persuadir a los demás agentes jugadores para que voten al mismo objetivo que nuestro agente. A continuación se muestran ejemplos de nuestra heurística de negociación el caso de pertenecer a ambos equipos.

Con tal de facilitar el funcionamiento de la heurística explicada a continuación se planean dos ejemplos diferentes. Para nuestro primer ejemplo vamos a asumir que tenemos dos agentes, nuestro DetectiveConan (Conan para abreviar), y otro agente diferente Bob. Estos agentes tienen asignados los roles de Seer y Possessed respectivamente. A continuación se muestra un ejemplo de diálogo en el cual Alice detecta una mentira lanzada por Bob y en respuesta lanza una *oferta* tal y como hemos descrito anteriormente:

Conan: DIVINED BOB POSSESSED
Bob: COMINGOUT BOB SEER
Conan: BECAUSE (AND (COMINGOUT BOB SEER) (DIVINED BOB POSSESSED)) (REQUEST(VOTE BOB))

Vamos ahora a modificar ligeramente las asunciones realizadas en el ejemplo previo. En este caso queremos ejemplificar el funcionamiento de la gestión de las mentiras *falsas*, por lo tanto nuestro agente DetectiveConan (Conan) será ahora el Possessed y Bob supondremos que juega con el rol Villager. En este caso el agente Bob sospecha que nuestro agente pueda pertenecer al equipo contrario, por lo tanto DetectiveConan actuará de la siguiente forma:

Bob: DIVINED CONAN POSSESSED
Conan: BECAUSE (DIVINED CONAN POSSESSED) (REQUEST(VOTE BOB))

De esta forma podemos observar con mayor claridad la gestión de ambos tipos de mentiras descritos en esta sección, así como las ofertas que propone nuestro agente negociador en respuesta a estas mentiras.

Como podemos observar, las ofertas lanzadas por nuestro agente durante la negociación consisten fundamentalmente en argumentos. Nuestro agente también esta preparado para entender y procesar argumentos con esta forma, teniendo en cuenta el marco argumentativo descrito en la sección anterior. Una vez finalizado el proceso de negociación el agente deberá escoger un agente como objetivo de su voto. Para ello se realizará el calculo de los argumentos aceptables [3] a partir del *famework* descrito en este trabajo. Debido a la naturaleza de este dominio solamente tendremos un único argumento aceptable (con mayor puntuación δ), cuyo *claim* consistirá en el objetivo de nuestro voto.

7. Normas de la competición de AIWolf para la ANAC2019

En la competición de la ANAC2019 se realizan en 2 tipos de partidas, de 5 y de 15 jugadores. En ambos casos, las reglas del juego no cambian, pero varía la distribución de roles entre los jugadores. En la Tabla 1 se muestra la distribución concreta de roles para cada modalidad de juego.

Tabla 1: Distribución de roles por modo de juego para la ANAC2019

	Modo	
Rol	5 jugadores	15 jugadores
Villager	2	8
Seer	1	1
Werewolf	1	3
Possessed	1	1
Medium	0	1
Bodyguard	0	1

La competición cuenta de dos fases: una fase clasificatoria y una fase final. La fase clasificatoria consta de una serie de rondas indeterminadas, que serán ejecutadas por los organizadores de la competición. Cada ronda procede de la siguiente manera:

1. Se selecciona un tipo de partida (5 o 15 jugadores) aleatoriamente.
2. Se seleccionan aleatoriamente los agentes que participaran en la partida.
3. Se realizan 100 partidas con los agentes seleccionados. En cada partida, se le asigna un rol aleatoriamente a cada agente.
4. Cada agente recibe 1 punto por cada partida en la que haya formado parte del equipo ganador.

Como en cada partida se asigna un rol aleatorio a cada agente, cada uno de los agentes enviados a la competición debe implementar estrategias para los 6 roles que existen en el juego del hombre lobo.

Los 15 agentes que haya recibido un mayor puntuación pasan a la fase final. En esta fase, se realizan una serie de rondas entre los agentes clasificados, que funcionan como en el caso anterior, y la clasificación final dependerá del número de puntos que obtiene cada agente en esta fase final.

Las condiciones de ejecución de los servidores de la ANAC2019 no permiten la lectura o escritura de ficheros por parte de lo agentes. Debido a esto, no ha sido posible integrar nuestro módulo de estimación de roles para esta competición. El módulo si que se puede utilizar de manera normal si se ejecuta en una máquina local, pero este hecho no nos ha permitido enviarlo para su evaluación frente a otros agentes inteligentes reales.

8. Resultados obtenidos por el agente *DetectiveConan*

Siguiendo las normas que se tendrán en cuenta en la ANAC2019, se han realizado una serie de experimentos con el agente implementado en este trabajo con el objetivo de evaluar su comportamiento. Puesto que es un juego de equipos, es complicado evaluar la calidad de un único agente por separado. Para este trabajo hemos considerado la realización de dos experimentos, un primer experimento con 15 agentes haciendo uso de los agentes de muestra proporcionados por los organizadores de la competición. Y otro experimento con 15 agentes, haciendo uso únicamente de nuestro modelo de agente.

En la Tabla 2 podemos observar los resultados del primer experimento. Compitiendo contra 14 de los agentes de muestra en una sucesión de 100 partidas con asignación de rol aleatoria, nuestro agente habría quedado en tercera posición. Analizando con mayor profundidad los resultados obtenidos en este experimento se puede observar como el porcentaje

	BG	MD	PS	SE	VL	WW	TOTAL
Sample3	1/6	1/6	13/14	2/9	7/46	17/19	0.410
Sample7	2/4	4/12	8/9	0/8	5/47	18/20	0.370
DetectiveConan	**0/7**	**0/5**	**9/13**	**0/4**	**8/50**	**19/21**	**0.360**
Sample1	1/7	0/5	4/6	2/6	6/50	23/26	0.360
Sample14	1/7	1/9	5/5	0/2	11/58	18/19	0.360
Sample2	1/5	1/3	4/4	1/8	10/60	19/20	0.360
Sample8	0/9	1/7	6/6	2/6	6/47	20/25	0.350
Sample12	1/8	1/4	6/6	1/6	8/56	17/20	0.340
Sample11	3/7	1/5	8/8	1/11	6/52	14/17	0.330
Sample4	0/4	1/7	5/8	2/8	7/55	17/18	0.320
Sample13	1/3	2/8	6/6	0/5	8/62	13/16	0.300
Sample9	1/7	0/6	3/4	1/9	8/55	16/19	0.290
Sample5	0/9	0/8	1/1	0/8	10/53	17/21	0.280
Sample6	1/9	0/6	2/4	2/5	6/56	17/20	0.280
Sample10	1/8	1/9	6/6	0/5	6/53	13/19	0.270

Tabla 2: Resultados obtenidos tras una serie de 100 partidas de 15 agentes haciendo uso de los agentes de muestra

	BG	MD	PS	SE	VL	WW
DetectiveConan	**20/100**	**20/100**	**80/100**	**20/100**	**160/800**	**240/300**

Tabla 3: Resultados obtenidos tras una serie de 100 partidas de 15 agentes haciendo uso únicamente del agente desarrollado en este trabajo

de partidas ganadas en el equipo de los hombres-lobo (64 %) es considerablemente superior al porcentaje de partidas ganadas en el equipo de los aldeanos (12 %) siendo el porcentaje de victorias total del 36 %. Es por ello que hemos considerado realizar el segundo experimento mencionado anteriormente. Con el objetivo de apreciar con mayor claridad este comportamiento se ha realizado un segundo experimento en el cual se han lanzado una sucesión de 100 partidas con asignación de rol aleatoria haciendo uso únicamente de nuestro agente. Los resultados de este segundo experimento se pueden observar en la Tabla 3. En este caso se observa como el agregado de los resultados obtenidos por los 15 agentes jugadores muestran un 80 % de victorias como jugadores del equipo de los hombres-lobo y un 33 % de victorias en el caso de jugar en el equipo aldeano.

9. Conclusiones

En este trabajo se ha explicado el diseño y el desarrollo de un agente negociador capaz de competir y cooperar en el juego del hombre lobo (*Werewolf*). En primer lugar se ha descrito el dominio, es decir, las normas fundamentales del juego en el que se desarrolla la competición. En segundo lugar se ha descrito el funcionamiento de la plataforma sobre la que se lanzan los agentes participantes así como los protocolos de comunicación entre agentes. Ya con toda la información fundamental expuesta, se ha presentado una propuesta para el modelado de oponentes haciendo uso de técnicas de aprendizaje automático. En esta sección se han podido observar resultados bastante positivos en lo que a predicción de roles de agentes oponentes se refiere. Además de realizar un modelado de oponentes, también se han presentado las técnicas de argumentación y negociación implementadas en el agente. Finalmente se han presentado brevemente las normas de la competición de la ANAC2019 y se han mostrado los resultados obtenidos en dos experimentos diferentes.

El agente desarrollado en este trabajo muestra un comportamiento muy bueno jugando en el equipo de los hombres-lobo y un comportamiento medio en el equipo de los aldeanos. Aunque debido a la naturaleza del juego estos pueden ser unos resultados un poco engañosos debido al factor de juego en equipo, se ha podido observar una mejora general respecto al agente de muestra proporcionado por los organizadores. Además, cabria esperar que formando equipo con agentes más inteligentes, mejore aún más el desempeño de nuestro nuestro agente en un dominio de estas características.

Referencias

[1] Trevor Bench-Capon. Value based argumentation frameworks. *arXiv preprint cs/0207059*, 2002.

[2] Trevor JM Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation*, 13(3):429–448, 2003.

[3] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.

[4] He He and Jordan L. Boyd-Graber. Opponent modeling in deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1804–1813, 2016.

[5] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1188–1196, 2014.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[7] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[8] Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. Modeling others using oneself in multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 4254–4263, 2018.

[9] Fujio Toriumi, Hirotaka Osawa, Michimasa Inaba, Daisuke Katagami, Kosuke Shinoda, and Hitoshi Matsubara. Ai wolf contest—development of game ai using collective intelligence—. In *Computer Games*, pages 101–115. Springer, 2016.