Diseño y desarrollo de un sistema argumentativo y una estrategia de negociación en un agente para la ANAC2019

Ramon Ruiz Dolz

April 2019

1. 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[1] 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 [4], 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.

1.1. 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.

1.1.1. 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 1.

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

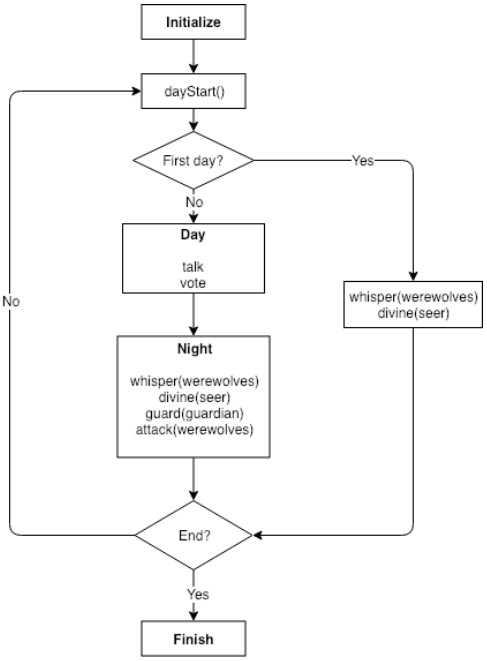

Figura 1: Diagrama de flujo de una partida en AIWolf.

1.1.2. 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 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 2, la asignación de roles viene dada por el servidor una vez la partida cumple los requisitos para ser iniciada.

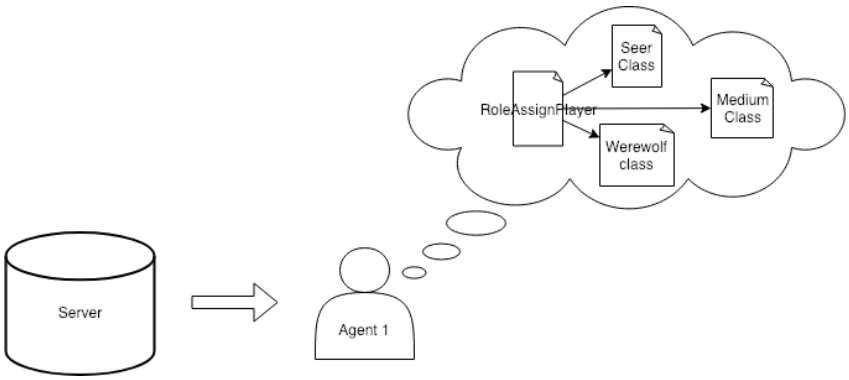

Figura 2: Asignación de roles a cada agente.

1.2. 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:

- **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.

2. 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 2.1 (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 2.2 (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.

2.1. 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 SAF tiene la siguiente forma:

- **Argumentos (A):**

  Tal y como ya se ha descrito en la Sección 1, la comunicación entre agentes sigue un protocolo predefinido. Es por esto que cada argumento existente en nuestro SAF 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 $\delta(\alpha, p_i)$ para un argumento emitido por el agente p_i tendrá un valor determinado en función del número de mentiras realizado por ese agente. Formalmente definimos esta función como,

  $$\delta(\alpha, p_i) = \frac{n_{pi}}{\sum_p^P n_p} \tag{1}$$

  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 SAF 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.

3. 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.

4. Resultados obtenidos por el agente *DetectiveConan*

En este trabajo se han realizado una serie de experimentos con el agente implementado 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.

	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

Cuadro 1: Resultados obtenidos tras una serie de 100 partidas de 15 agentes haciendo uso de los agentes de muestra

En la Tabla 1 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 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 2. 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.

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

Cuadro 2: Resultados obtenidos tras una serie de 100 partidas de 15 agentes haciendo uso únicamente del agente desarrollado en este trabajo

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] 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.