3rd International AIWolf Competition Regulation[1]

Ver 1.2.2 (2021)

These rules describe the Werewolf game rules used in the contest, the contest evaluation rules (how a winner is chosen), and the submission rules. These rules concern the “Protocol” task. For the NLP task, please see the NLP task regulation.

1. Werewolf Game Rules

1.1 Game Size and Role Distribution

An AIWolf game on the contest server can be composed of 15 players or 5 players. The role distribution depends on the size of the game, as follows:

- 15 player game:
  - 11 Villager-aligned players (8 Villagers, 1 Seer, 1 Medium, 1 Bodyguard)
  - 4 Werewolf-aligned players (3 Werewolves, 1 Possessed)
- 5 player game:
  - 3 Villager-aligned players (2 Villagers, 1 Seer)
  - 2 Werewolf-aligned players (1 Werewolf, 1 Possessed)

1.2 Role Description

A player can be villager-aligned or werewolf aligned. There are 6 possible roles.

1.2.1 Villager-aligned roles (4 types):

Villager: Has no special abilities.

Seer: At the end of each day, the seer can choose one player to “Divine.” The seer will know whether this player is a werewolf or a human.

Medium: When a player is eliminated from the game by voting, the Medium is informed of whether that player was a werewolf or a human. If there is no Medium in the game, this information is not revealed for eliminated players until the end of the game.

1 Seventh AIWolf Competition

Bodyguard: At the end of each day, the bodyguard can choose one other player to “Guard.” That player is not affected by the attack of werewolves. The bodyguard does not receive any feedback about the result of this action (i.e., whether they successfully protected an attack or if the werewolves did not attack).

1.2.2 Werewolf-aligned roles (2 types):

Werewolf: At the end of each day (except the first), the werewolves vote on a human player. The player with the highest number of votes is Attacked. Also, only the werewolves can use the “whisper” channel to exchange information while hidden from human players.

Possessed: The Possessed has no abilities, like the Villager. However, the Possessed wins with the Werewolf team. Seers and Mediums identify the Possessed as human. Also, the possessed can NOT use the "whisper" channel.

1.3 Talk Rules (Conversation between agents)

1.3.1 Turn system

Conversation happens in turns. Each player can broadcast one message at each turn. The order in which the players broadcast within a turn is random.[2] A player may choose not to broadcast a message during a turn by using the “Skip” message (which indicates the player wants to talk later in this talk phase) or “Over” message (which indicates the player does not want to talk anymore in this talk phase).

In a day, each player may broadcast up to 10 messages. This limit does not include "Skip" and "Over" messages.

The "talk" phase ends when: (a) every player broadcasts "Over"; (b) or when every player broadcasts "Skip" three times in a row; (c) when 20 turns have passed.

1.3.2 Conversation on the first day

On the first day of the game (day 0), the talk phase is skipped, and no conversation takes place.

1.3.3 Werewolf Whisper

Each day, the werewolves can use the “whisper” channel after the end of the day phase (after the elimination vote), before the attack vote.

In the first day (day 0), although there is no talk phase, and also no attack vote, the whisper channel can still be used. The werewolves can use initial “whisper” phase to discuss strategies such as fake role claims for day 1.

2 Note: Until 2019, message exchange within a turn was synchronous: all the agents submitted the turn's message at the same time, and all messages were revealed at the same time. From 2020, the message exchange is not synchronous. Each agent receives all the messages sent up until their time to broadcast. Please be careful of the difference when using old code.

If there is only one werewolf in the game, no whisper takes place. Please be careful that the whisper method is not called in this situation.

1.4 Voting Rules

Every day, except for the first day (day 0), a vote to eliminate a player from the game happens after the “talk” phase. The server informs the werewolves, seers, and bodyguards about which player was eliminated before asking for their special action, so they can use this information to make their decision. Other players are informed about which player was eliminated on the beginning of the following day.

If there is a tie, the elimination vote is repeated one time. In this case, there is no extra conversation. If there is still a tie after the second vote, then the eliminated player is chosen randomly from the tied players. During a revote, all players have a vote, and they can vote in anyone, even players that were not voted on in the first vote.

The werewolf attack vote happens in the same manner as the attack vote. One revote is allowed in case of a tie, and there is no whisper before the revote.

1.5 Special Abilities

1.5.1 Seer

During the night phase, the Seer can choose a player. The seer receives information about whether that player is a human or a werewolf. The Seer receives the name of the player eliminated by voting before choosing the divination target. It is possible to do divination on the first day (day 0).

1.5.2 Medium

The Medium receives information about whether the player eliminated by voting was a human or a werewolf. Note that there is no vote on the first day (day 0), so the Medium will receive no information that day.

1.5.3 Bodyguard

The bodyguard can choose any player other than themselves to protect. That player is not affected by the werewolves’ attack. The bodyguard can choose a player that has been eliminated, but nothing happens in that case. The bodyguard receives the name of the player eliminated by voting before choosing the protection target.

2. Preliminary Contest and Final Contest

2.1 Game Set:

In this competition, one “Game Set” is the challenge unit, and defined as a set of 100 games among a fixed grop of players. One “Game Set” is executed as follows:

1. From the teams registered in the competition, 5 or 15 agents are selected randomly to form the “village”.
2. Roles are assigned to the members of the village randomly, and one game is executed.
3. Step 2 is repeated 100 times.

An instance of each agent is created at the beginning of the Game Set, and destroyed at the end of the Game Set. In other words, the instance of the agent is kept through the 100 games in the set, and information can be kept from one game to the next inside the instance’s memory. Additionally, the agent ID (AgentIdx) of each instance is kept inside the Game Set.

2.2 Testing Contest:

Before the Preliminary Contest and the Final Contest, each registered contestant can participate in the Testing Contest. When the registered team submits an agent to the testing server, the agent will participate in the testing contest a few times every day. In the Testing Contest, each team will participate in at least one Game Set, to check if there are any errors. Each contestant can download the logs of their own agent. Please use the Testing Contest to check if your agent is working correctly, and make any fixes or improvements necessary.

2.3 Preliminary Contest:

After the submission deadline closes, we will perform the Preliminary Contest between all registered participants that submitted at least one working revision. In the Preliminary Contest, each registered participant will participate in a minimum number of Game Sets. The 15 contestants with the highest winning rate are selected to be finalists.

2.4 Final Contest:

The 15 finalists selected in the preliminary contest will be invited to participate in the Final Contest. They will have some time after the finalist announcement to optionally submit a final revision of their agent.

The final contest will consist of at least one more Game Set (possibly more, depending on the computational resources available). The final winner will be selected by the winning ratio of all games in the final contest.

3. Agent Submission (How to submit an agent)

3.1 Team Registration:

To register a team in this competition, you must make an account in the competition webpage: http://contest.aiwolf.org/. After you create your account, you can register for the competition.

3.2 Player Submission:

Submit the files that implement an agent capable of playing all roles described in section 1.1 of this document. The files to be submitted depend on the programming language used.

3.2.1 Java Agent

File to submit: jar archive

For teams creating a Java agent, you may submit a single jar file containing all your code. If you are using libraries for machine learning or other things, please include them in your jar file. It will automatically create the necessary classpaths.

Also, include any data files that you may need in the jar archive, and read them from there. For example, if you need to read the file /data/foo.txt inside the jar archive, you can use *"InputStream is = getClass().getClassLoader().getResourceAsStream("data.foo.txt")"* to create an input stream that reads your data.

Please be aware that there may be conflicts if you include the following files in your jar file "aiwolf-client.jar", "aiwolf-server.jar", "aiwolf-common.jar", "aiwolf-viewer.jar", "jsonic-xxxx.jar". This may cause problems when running your agent. So avoid including files that use these names in your jar file.

The recommended Java version is 11. If you use versions other than that, your agent may not run correctly. Please test your agent in the test server.

3.2.2 C# Agent

File to submit: dll file, zip archive

If you create a single dll file, please register your dll file directly. If you create multiple dll files, please submit all of them as a single zip archive. In this case, there is no restriction for file names.

However, if you submit a zip archive, make sure to name the file with your player class as "teamname.dll". For example, if your team name is "tori", your player class needs to be inside a file named "tori.dll", or it will not run. If you are including a single dll, there are no restrictions on the filename.

If you embed a data file in the assembly as an embedded resource, you can get a Stream with

the data using the method Assembly.GetManifestResourceStream.

3.2.3 Python Agent

File to submit: zip archive

When submitting your agent, you must specify the name of the agent script. Also, you must include the agent script inside a directory with your team name. For example, if your team is registered as “team_foo”, and your start script is indicated as “start_bar.py”, your zip file should contain the path “team_foo/start_bar.py”. In the submission form, remember that your start script should only be “start_bar.py”, without the directory name.

Additionally, any other scripts that the agent executes should also be included in the “teamname” directory (“team_foo/”). Scripts included in other directory may not run, or cause the agent to crash.

3.3 Source Code and Algorithm Description

Agents selected to the finalist tournament must submit their source code. When submitting the finalist version of your agent, you will be instructed to submit the source code of all files used. The exact method of submission will be detailed after the preliminary contest. Agents submitted without source code will be disqualified.

Additionally, the finalists are also requested to submit a short document that describes the key ideas of their submitted agent.

4. Forbidden activities for Agents

During the competition, the following activities are forbidden to the agents. Agents that do not follow these restrictions, or agents that throw errors during the contest, will be eliminated. In certain cases the organizers might contact the team directly to find a solution, at the organizer’s discretion.

- Broadcasting a message that does not follow the protocol (that cannot be created by the ContentBuilder)
- Writing data to files (reading files is permitted under certain conditions)
- Connecting to a network
- Creating new threads
- Executing a program as a separate process
- Taking more than 100ms to respond to a request from the server (small deviations from this time limit might be tolerated)

About reading data from files: Your agent may only read data from the files that you have uploaded when registering to the contest. Java agents may read data from the .jar file that was registered. .NET agents may read data from the .dll files submitted. Python agents may read data from the files included in the .zip archive that was submitted.

5. About running player programs

In this competition, we will use the aiwolf-ver0.6.2 server available at http://aiwolf.org/en/server. We will use the instructions listed under “Running AIWolf Server” in the same webpage.

To create a java agent that can join the game, prepare a java program that inherits the *org.aiwolf.common.data.Player* Interface inside *AIWolfCommon.jar*. Or prepare a program in .NET or Python that can communicate with the server in the same manner as the Interface above.

5.1 Methods that need to be implemented in the Player Interface (Java)

A class inheriting the Player interface needs to implements 11 methods. These methods are divided in the following 4 groups:

- Methods for organizing information: *initialize, update, dayStart, finish*
- Methods for targeted actions: *vote, attack, guard, divine*
- Methods for dialogue: *talk, whisper*
- Methods for naming: *getName*

5.1.1 Methods for organizing information (initialize, update, dayStart, finish)

These methods are used to process information, and do not need to return anything.

initialize(GameInfo, GameSetting): This method is called once at the start of the game. The arguments are the current state of the game *GameInfo*, as well as the information about the setup of the game (number of players and roles) *GameSetting*.

update(GameInfo): Is called before each invocation of all other methods (except *initialize*), to update the agent’s information about the game state. When called before *finish* it will also provide the full list of players and their roles.

dayStart(): It is called once before each day phase.

finish(): It is called after the game is finished.

5.1.2. Methods for targeted actions (vote, attack, guard, divine)

Methods that must return the ID of the agent to be targeted by the action. In the case of *Attack, Guard, Divine*, these methods will only be called on agents with the respective roles.

vote(): Return the player to be voted out of the game on that day phase.
attack(): Only for werewolves. Return the player to be voted for attack on that night phase.
guard(): Only for the bodyguard. Return the player to be protected on that night phase.
divine(): Only for the seer. Return the player to be investigated on that night phase.

5.1.3. Methods for dialogue (talk, whisper)

These methods must return the message to be broadcast (in String format). *whisper* is only

called for werewolf players.

talk(): Returns a message that will be broadcast for all active players. In this competition, the message must be constructed using the class org.aiwolf.client.lib.ContentBuilder, which will guarantee that it follows the defined protocol. (See section 5.3)

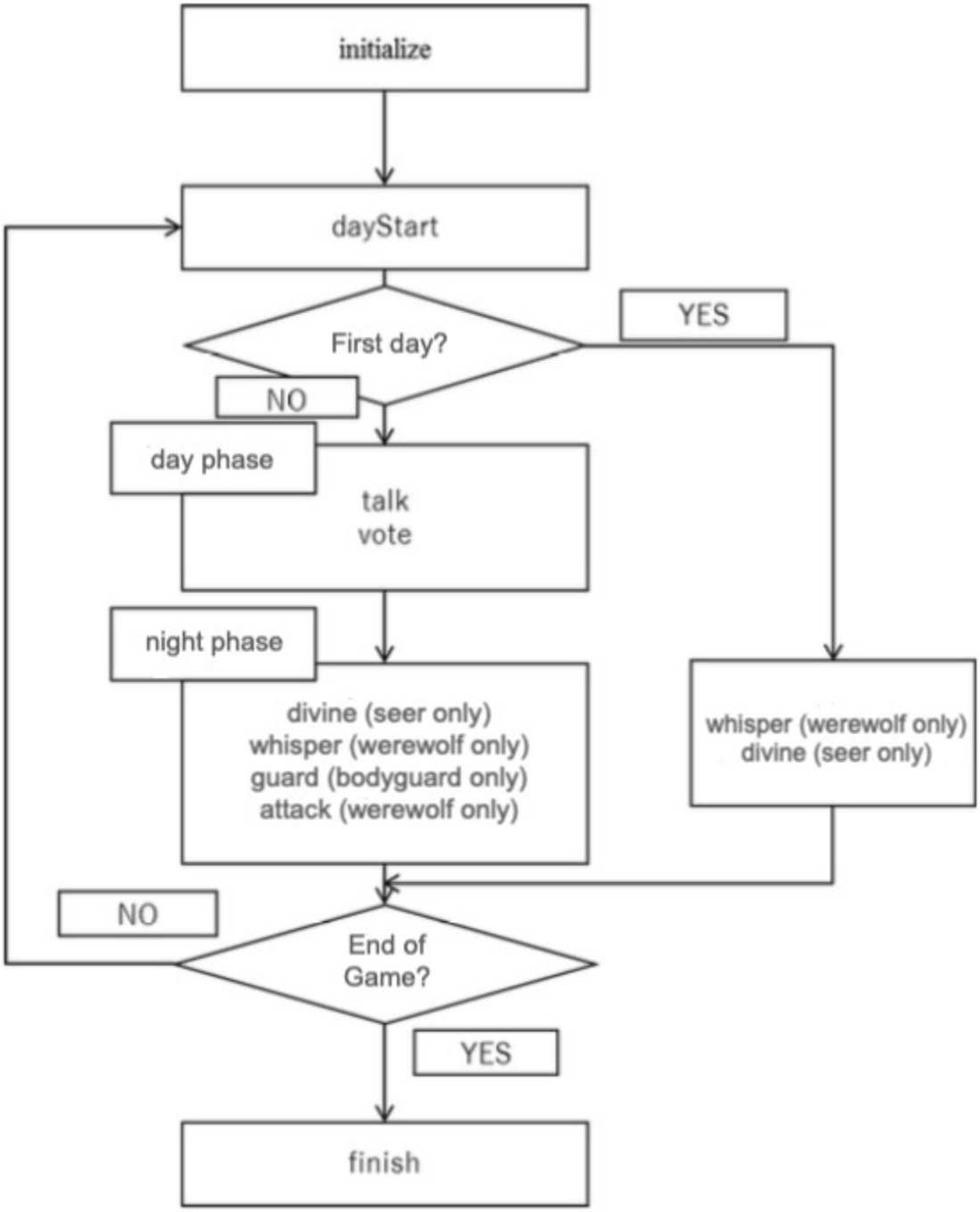
whisper(): This method is only called for werewolf agents. The messages returned for this method are not displayed to the non-werewolf players. In this competition, the message must be constructed using the class org.aiwolf.client.lib.ContentBuilder, which will guarantee that it follows the defined protocol. (See section 5.3)

5.1.4. Naming Methods (getName)

getName(): Returns the name of the player (in String format). The name of the player is displayed on the Game's log. Please return the team name that you registered during your registration for the competition. If you return a different name, this may result in disclassification.

5.2. Timing for invoking each method.

The methods described in the previous section (except for getName) are called following the flow described below. While the flow does not include the "update" method, "update" is called before every other method, with the exception of initialize.



5.3. Valid Broadcast Strings

In this competition, all dialogue among agents must be composed of strings that can be created by the org.aiwofl.client.lib.ContentBuilder class. There are 23 kinds of messages, listed below. Please see the “Protocol Version 3.6” documents for more details on the protocol.

- estimate：X believes that Y’s role is Z. (EstimateContentBuilder)

- comingout：X declares that Y's role is Z.(ComingoutContentBuilder)
- divination：X divines Y. (DivinationContentBuilder)
- divined：X divined Y, and the result was Z (Human or Wolf) (DivinedResultContentBuilder)
- identified：X used the medium power, and identified Y as Z (Human or Wolf) (IdentContentBuilder)
- guard：X protects Y. (GuardCandidateContentBuilder)
- guarded：X protected Y. (GuardedAgentContentBuilder)
- vote：X votes for Y. (VoteContentBuilder)
- voted：X voted for Y. (VotedContentBuilder)
- attack：X votes for Y to be attacked.（AttackContentBuilder）
- attacked：X attacked Y. (AttackedContentBuilder)
- agree：X agrees with broadcast T. (AgreeContentBuilder)
- disagree：X disagrees with broadcast T. (DisagreeContentBuilder)
- request：X requests Y to do Z (RequestContentBuilder)
- inquire：X inquires Y about Z (InquiryContentBuilder)
- because：X states sentence Z because of reason Y (BecauseContentBuilder)
- and：X states A and B and ... (AndContentBuilder)
- or：X states at least one of A or B or ... (OrContentBuilder)
- xor：X states either A or B (XorContentBuilder)
- not：X negates Y (NotContentBuilder)
- day：X states that Y happened on day T (DayContentBuilder)
- over：I have nothing else to say (if all players broadcast OVER, the Talk Phase ends)(OverContentBuilder)
- skip：I will say nothing now (even if all other players broad cast OVER, the Talk Phase does not end) (SkipContentBuilder)

5.4. About the Player class package

Please create an unique Player class. Avoid just rewriting the sample classes such as org.aiwolf.sample.player.SampleRoleAssignPlayer.

Also, please include your Player Class in an independent package. The recommended package naming convention is to use the reverse order of your email's address. (For example, if your e-mail address is contestant@example.com, and your Player class is MyPlayer, the header of your class would probably look like this:

```
package com.example.contestant;
Import org.aiwolf.sample.lib.AbstractRoleAssignPlayer;

public class MyPlayer extends AbstractRoleAssignPlayer {

}
```

In this case, when submitting your source code, you should write the following in the “class path” section:

```
com.example.contestant.MyPlayer
```

6. Using other Programming Languages

Besides Java, we will consider entries in Python and .NET. When using these languages, please check their respective libraries. Using TCP-IP libraries for socket communication, and making sure that the correct messages are passed back and forth between server and client, it is possible to participate in the game. When in doubt, please contact the organizing committee: gm@googlegroups.com

However, please note that the contest will be executed in a linux machine, so if you do your development in a different environment, make sure that your code is portable. Clients that cannot run on the server environment will be automatically disqualified. For example, in a previous contest one player used a system-dependent JSON library for python that did not run on the linux server, and caused disqualification. Please be careful.

7. Changes

These regulations are subject to change at any time. Changes will be announced at the project page (http://aiwolf.org/en), the development mailing list (aiwolfdev@googlegroups.com) or our Slack Channel (https://aiwolfen.slack.com).

8. Change History

2019/02/09 -- ver 1.0.0 -- Translated from the 4th AI Wolf competition
2019/05/09 -- ver 1.1.0 -- Clarification about the practice contest, clarification about reading from files, other minor changes.
2020/03/29 -- ver 1.2.0 -- Updated rules for 2020 (non-synchronous broadcast, Java 11)
2021/03/22 – ver 1.2.1 – Updated rules for 2021 (minor language improvement, no rule changes)
2021/04/02 – ver 1.2.2 – Updated description of the contest executrion (section 2) and submission files (section 3) (No rule change, text improvement)