txwerewolves Documentation

Carl Waldbieser

Feb 25, 2023

Contents:

1 Werewolves! 1
- 1.1 Deployment Considerations . . . 1

2 Install, Configure, and Run the Game 3
- 2.1 Installing the Software . . . 3
- 2.2 SSH Service Configuration . . . 3
- 2.3 Web Service Configuration . . . 4
- 2.4 Starting the Service . . . 4

3 Playing the Game 5
- 3.1 Client Behavior . . . 5
- 3.2 The Lobby . . . 5
- 3.3 The Werewolves! Game . . . 5
- 3.4 The Chat Feature . . . 6
- 3.5 Victory Conditions . . . 6
- 3.6 Roles . . . 6

4 Glossary 9

5 Technical Documentation 11
- 5.1 Architecture . . . 11
- 5.2 The Terminal Service . . . 11
- 5.3 The Web Service . . . 12
- 5.4 Users . . . 14
- 5.5 Sessions . . . 14
- 5.6 Avatars . . . 14
- 5.7 Applications . . . 14

6 Indices and tables 15

Index 17

CHAPTER 1

Werewolves!

Werewolves! is a party game sometimes called “Mafia”. There are many variations. Werewolves! is a computer network based xgame based adaption of the game.

The players are randomly assigned roles of either villagers or werewolves. 3 additional roles are dealt to the table. At night, some of the villagers with special roles and all the werewolves wake up at different times and perform certain actions. After all the night phases are done, the players may discuss what happend during the night, though they may not actually show each other their cards. A player may say the role she received, but she may also claim that she were dealt a role that she did not receive.

After the discussion, each of the players casts a vote to eliminate a player. If there is a player or players who received more than 1 vote, the players with the most votes are eliminated and the actual roles the players now possess are revealed. If a werewolf was killed, the village team wins, even if an innocent villager was eliminated, too. The villagers may also win if no one is eliminated and all the werewolves are in the middle of the table.

Some roles have specialized winning conditions (e.g. the tanner).

In this adaption, players interact with each other via terminal or web clients. The game shows players only partial information and allows them to take actions based on the roles they were dealt during the night phase.

1.1 Deployment Considerations

A basic deployment scenario for this game is on a home network, and players can connect to the service via their smart phones, tablets, laptops, etc. The game is lightweight enough that it can be run on a laptop computer that has its network adapter in “hotspot” mode, and clients can connect directly to it via wifi.

The game does feature a simple in-game chat so that it can be played over a wider area where players are not in each other’s physical presence, but the networking concepts involved in more complex setups are not covered in these instructions. The game could also be hosted on a cloud platform that players connect to, but again, such advanced deployments are not covered here.

CHAPTER 2

Install, Configure, and Run the Game

The service supports both authenticated SSH terminal clients as well as unauthenticated browser-based web clients. A single service supports both types of clients.

2.1 Installing the Software

2.1.1 Installing from the Python Package Index (PyPi)

txwerewolves can be installed from the Python Package Index. Althought not specifically required, I *strongly* recommend installing into a Python virtual environment. I like to use virtualenvwrapper .

```
$ mkvirtualenv wolfenv
(wolfenv)$ pip install txwerewolves
```

2.1.2 Installing from cloned Github repo

Clone the source repository and install the dependencies via *pip* into a Python virtualenv.

E.g.

```
$ git clone https://github.com/cwaldbieser/txwerewolves.git
$ cd txwerewolves
$ pipenv install
```

The last command uses *pipenv* to install the dependencies in `Pipfile.lock`. This command can fail if operating system dependencies are not satisfied. Satisfying those dependencies isn't covered here.

2.2 SSH Service Configuration

To set up the SSH service you need to create a randomly generated SSH key pair:

```
$ mkdir -p ~/.txwerewolvesrc/ssh_keys
$ ckeygen -t rsa -f ~/.txwerewolvesrc/ssh_keys/ssh_host_rsa_key
```

To configure authentication for the SSH service, edit *~/.txwerewolvesrc/users/user_keys.json*.

```
{
    "user1": [
        "pubkey ...",
        "another pubkey ..."
    ],
    "user2": [ "only one pubkey ..." ],
    "user3": [ "etc ..." ]
}
```

Note: The location of the SSH service keys and user database can be configured from the command line. The service will also check in *$HOME/.txwerewolvesrc* and */etc/txwerewolves* for the sub-directories *users* and *ssh_keys*.

2.3 Web Service Configuration

The web service doesn't support any advance configuration options at this time.

2.4 Starting the Service

To start the service:

```
(wolfenv)$ twistd -n werewolves
```

When running against a cloned git repo, you need to add the project folder to your PYTHONPATH.

```
$ cd /path/to/project
$ export PYTHONPATH=.
$ twistd -n werewolves
```

To connect an SSH client to the service (assuming a typical OpenSSH command-line client):

```
$ ssh user1@localhost -p 2022
```

To connect a web client to the service, simply browse to the IP address of the interface and the port on which the web service runs. E.g. http://192.168.0.100:8080/

CHAPTER 3

Playing the Game

3.1 Client Behavior

The game can be played using a terminal client or a web client. A player may switch freely between clients, but only one will remain open for a given user ID at any one time. If a player logs into the service with another client (web or terminal), the previous client is automatically disconnected from the service.

It is possible to disconnect from the service without quitting the game. The CTRL-D key combination in the terminal client will do this. There is no such explicit command for the web client, but the player can simply close her browser and later log in with a new client to resume where she left off.

3.2 The Lobby

When you first log into the service, you will be in the lobby. The menus will allow you to start a session or join a session that another player has started.

A player who creates a session may choose to start a session. Once a session starts, all players who accepted invitations to the session are placed into a game while any pending invitations are revoked.

3.3 The Werewolves! Game

During gameplay, the menus and status areas will guide you through the game. The player information area shows your player name and the role you were randomly assigned. This information is only available to you.

The game information area shows all the roles that will be used in this game. The session owner can adjust these settings and restart the game with the new settings.

> **Warning:** Changing settings will restart a game in progress!

General game information is available to all players throughout the game. Note that there are always 3 more roles than there are players. These roles are assigned to the "table". Players will not know for certain which roles are actually held by the other players in the game and which are "on the table."

The phase information area tells you information about the current phase of the game. It will provide a brief overview of the phase and prompt you advance to the next phase. All players must indicate they are ready to advance before the game will advance to the next phase. Until then, a status indicator will note that the game is waiting on other players. The information in this area is mostly the same for all players, though additional information may be provided to specific roles.

The role power area allows specific roles to exercise their powers during the appropriate night phases. During the Daybreak phase, all players will be allowed to vote for a single player to eliminate. If any player receives more than a single vote, then the player or players with the most votes is eliminated.

At the end of the game, post game results are displayed. Who voted to eliminate whom, who was eliminated, who was dealt what role, and what role each player actually ended up with are revealed to all players. The winning team is also announced.

3.4 The Chat Feature

Players may chat with each other in the lobby and during the *Werewolves!* game using a community chat window.

It is against the spirit of the rules to discuss your actual role until the Daybreak phase. Once the Daybreak phase has been reached, the chat feature is an important way to try to figure out what happened during the night phases, unless all players are able to communicate freely by other means (e.g. they may all be playing together in the same room).

Players may adopt strategies of telling half-truths or outright fibs in order to ferret out the truth of what really happened.

3.5 Victory Conditions

The werewolf team wins if at least one player is a werewolf AND no werewolves were eliminated.

The village team wins if at least 1 werewolf was eliminated OR no one was eliminated and no player was a werewolf.

The tanner wins only if the player who holds this role at the end of the game is eliminated. The village team can win a joint victory with the tanner if a werewolf is eliminated in addition to the tanner.

The minion wins with the werewolf team, even if the minion is eliminated. If no players are werewolves but 1 player is the minion, the werewolf team can still win if a member of the village other than the tanner or the minion is eliminated.

3.6 Roles

- Villager - no special powers, wins with the Village team.
- Seer - Can use her mystic powers to either view 2 of the 3 roles on the table or 1 player's role. Wins with the Village team.
- Robber - May steal a role from another player. That player gets robber role. The robber gets to see his new role. The player with the robber role at the end of the game wins with the Village team.
- Troublemaker - The troublemaker may choose to swap the roles of 2 other players *without* looking at them. The troublemaker wins with the Village team.
- Insomniac - The insomniac wakes up at the end of the night and checks to see if her role changed. The player with the insomniac role at the end of the game wins with the Village team.

- Werewolf - All the werewolf players wake up together at night and can see each other. A player holding a werewolf role at the end of the game wins with the Werewolf team.
- Minion - The minion wakes up after the werewolves and can see who they are. The werewolves *cannot* see who the minion is. The player holding the minion role at the end of the game wins with the Werewolf team. Note that the Werewolf team wins even if the minion is eliminated, such is his fanaticism.
- Tanner - The tanner has a profession that has left him longing for the sweet embrace of death. The tanner only wins if he is eliminated. The Werewolf team does *not* win if the tanner is eliminated, because a good deed will have been done for this poor soul. The Village team does not win if the tanner is eliminated alone (his blood is on the villager's hands, after all), but the village can win a joint victory with the tanner if a werewolf is eliminated with him– the vanquishing of a cursed one allows for some collateral casulties.

CHAPTER 4

Glossary

werewolf: A cursed individual who assumes the form of wolf-like creature at night, especially during the full moon. Werewolves prey on the living, and they possess great ferocity and strength in their cursed forms. They are vulnerable in their human forms during the daylight hours.

CHAPTER 5

Technical Documentation

5.1 Architecture

The game runs as a single process which is launched via the Twisted command line program, **`twistd`**. The application is written as a Twisted Application Plugin.

The main entry point is *serviceplugin.py*, which will be located in the `twisted/plugins` folder of your source repository or of your Python environment's `site-packages` folder, depending on whether you have cloned the repository or have installed the software (e.g. with **`pip`**).

There is a global name, *serviceMaker* at the end of this script which is bound to an instance of a class that implements the `twisted.application.service.IServiceMaker` interface. When **`twistd`** runs, it detects this application plugin and displays its *tapname* attribute as one of its subcommands. If you run the *werewolves* sub-command, the `makeService()` method is called on this instance. This is the program entry point.

The `makeService()` method parses command line options, looks for configuration files, and generally configures both the terminal service and the web service provided by the game. The two services are set as child services of a generic parent `MultiService()` instance. The general Twisted reactor framework will begin to send network events to the services as they become available.

5.2 The Terminal Service

The terminal service is implemented by the class `txwerewolves.service.SSHService`. The service is initialized by the `txwerewolves.service.SSHService.startService()` method which overrides `twisted.application.service.startService()`. A number of components need to be created to start and configure this service. Because this service uses SSH private-key / public-key authentication, the secret and public materials for the service must be read in from the file system. An instance of `twisted.conch.ssh.factory.SSHFactory` is created and configured with the key material. The factory will be responsible for creating an SSH protocol instance when a client connects to the service. We'll see how this is configured later on when the endpoint for the service is created.

Because the service is authenticated, the Twisted Cred framework is used to provide authentication services. This means the service will require a portal, a realm, and at least one credential checker. The portal is a simple instance

of twisted.cred.portal.Portal. The realm is an instance of txwerewolves.auth.SSHRealm. The portal is initialized with this realm.

The final component need for the authentication system is a credential checker. Fortunately, Twisted provides twisted.conch.checkers.SSHPublicKeyChecker. This checker needs to be initialized with a public key database. The game uses twisted.conch.checkers.InMemorySSHKeyDB to fill this need. The keys are read in from a simple JSON file of usernames and their public keys and stored in the database. The credential checker is contructed from the key database and it is registered with the portal. The portal is assigned to a property of the SSHFactory instance.

Finally, a Twisted endpoint is constructed from a string description and instructed to listen and use the factory to produce a protocol that will communicate with the connected client. Once a connection is made, the SSHFactory creates an instance of twisted.conch.ssh.transport.SSHServerTransport to communicate with the connected client.

5.2.1 The Terminal Realm and Avatar

When a client connects to the terminal service and successfully authenticates with the registered credential checker, the resulting avatar ID is passed to the SSHRealm instance that the portal was initialized with. The SSHServerTransport also requests the twisted.conch.interfaces.IConchUser interface[1], and it expects the realm to return an avatar that supports this interface.

The SSHRealm instance takes the avatar ID and registers it as in the game's user database using txwerewolves.users.register_user(). If a current avatar exists for the user, it is shut down and a new txwerewolves.auth.SSHAvatar is created to replace it. The active avatar for the user is stored in the user database as a property of the user entry.

The new avatar is a subclass of twisted.conch.avatar.ConchUser, so it inherits much of the code required to communicate with a client terminal. Namely, it's openShell() method will be called when the client requests a shell. The avatar will use a slightly modified twisted.conch.insults.insults.ServerProtocol instance to connect a txwerewolves.term.TerminalAdapterProtocol to the SSH protocol connected to the client terminal. The TerminalAdapterProtocol is a subclass of twisted.conch.insults.insults.TerminalProtocol so basic curses-style abstractions are available to the application code.

The terminal avatar delegates many of its functions to its terminal adapter. Initially, the avatar installs the default terminal application as a property of the user entry in the user database. The terminal adapter is also responsible for translating user input from the client into events that can be handled by the application protocol.

5.3 The Web Service

The web service for the game is provided by txwerewolves.webservice.WebService. The service setup is similar to the basic setup for the *terminal service*. The service entry point is txwerewolves.webservice.WebService.startService(). A web site is created and a Twisted endpoint is constructed from string description of the host, port, and connection options. The endpoint is made to listen for incoming network events. It is configured to use the web site instance to create a protocol that will communicate with the web client.

5.3.1 The Web Site

The web site is created using the Klein micro-framework. While the web client allows anyone to log in with a self asserted username, it still uses the Twisted Cred system to "authenticate" players. Before constructing the web site, a txwerewolves.webauth.WebRealm is created. A portal is created and initialized with the realm. An

[1] Actually, this is rather simplified. SSHServerTransport actually calls on twisted.conch.ssh.userauth.SSHUserAuthServer to perform the user authentication.

instance of `txwerewolves.webauth.WerewolfWebCredChecker` is registered as the credential checker for the service.

An instance of `txwerewolves.webservice.WebResources` is created and will serve as a model for the resources in the web site. The root resources is obtained from this instance, and a `twisted.web.server.Site` is initialized with this root resource. The site will act as factory that creates protocol instances that interact with incoming HTTP requests.

5.3.2 Resources

HTTP requests are routed to resources in the web application. The resources have the following meanings:

- **/login** - If the client browser has not established a session, it will be redirected to this resource. Here, a player may choose and submit a login name to establish a session.
- **/logout** - A client that accesses this resource will have its session expired.
- **/static/** - Static resources like JavaScript, stylesheets, etc. are served from this resource tree.
- **/action** - Clients can POST the action a player chooses to this resource to send commands to the application.
- **/settings** - Clients can POST settings to this resource to change application settings and reset the application.
- **/chat** - Clients can POST chat messages to this resource that will be displayed in community chat dialogs.
- **/werewolves** - This resource serves the static HTML for the Werewolves! game.
- **/werewolves/...** - There 6 distinct sub-resources a client can request: *actions*, *phase-info*, *player-info*, *game-info*, *output*, and *request-all*. Requesting one of these resources will trigger a corresponding update to be sent via a Server Sent Event (see the */subscribe* resource, below).
- **/lobby** - This resource serves the static HTML for the Lobby application.
- **/lobby/...** - There are 2 sub-resources, *status* and *actions*. Requesting one of of these resources will trigger a corresponding update to be sent via a Server Sent Event (see the */subscribe* resource, below).
- **/subscribe** - Clients can request this resource to subscribe to server sent events from the application. These events are received by JavaScript handlers in the client browser that can update the user interface with new information or actions to be selected.

5.3.3 The Web Avatar

The web avatar is is created when a player browses to the */login* resource of the web service. It triggers any existing avatar to shut itself down. The new avatar replaces the old avatar in the user database for the appropriate user entry. Any existing application will be converted to a similar application of the appropriate kind (e.g. a terminal Lobby application will be converted to a terminal web application).

Unlike the *web avatar*, the web avatar either handles its own requests or forwards them directly to the currently installed web application.

5.3.4 Rich Client Interface

Unlike, the *terminal service* client, the web client is meant to be a web browser. A web browser has a rich client interface, including its own embedded scripting engine. Javascript event handlers are included in the web pages to handle player input and to provide updates from the web service's */subscribe* resource. Contrast this with the terminal client, which provides a simple interface for producing character output at specific coordinates.

5.4 Users

5.4.1 The User Database

The user database is a mapping of user entries that have been registered in the service, either via the *terminal service* or via the *web service*. Each player is identifier by a unique avatar ID which maps to a corresponding user entry.

5.4.2 User Entries

A user entry contains the avatar ID of the player. It also contains a reference to the current avatar for the player and a reference to the current application. The entry also has some basic state information concerning whether a player has been invited to or has joined a session.

5.5 Sessions

5.6 Avatars

5.7 Applications

The user entry for a given player always has a single active application registered at any given time. The application must be the appropriate kind for the current user avatar. E.g. terminal avatars require terminal applications and web avatars require web applications. Applications must be convertible from one kind to another so that a user can switch between interfaces. Because of this, an application should maintain state in its *appstate* property which can readilly be transferred to a similar application of a different kind. Applications implement the `txwerewolves.interfaces.IApplication.produce_compatible_application()` to perform such conversions.

5.7.1 The Lobby Application

The Lobby application is the default application. Each player's state is maintained by a finite state machine. Players may move through various states. A player may start a session and invite others to join. She may accept an invitation or reject it. She can leave a session or cancel a session she initiated. She may also start a session which players have joined. this latter action will replace the Lobby application for players who have joined the session with the Werewolves! game application.

5.7.2 The Werewolves! Application

The Werewolves! game application is different from the Lobby application in that the game state is shared amongst the members of the session. In contrast, the Lobby application of a given player doesn't share state with the Lobby applications of other players– all interaction is mediated via changes to a shared session instance.

The game application is modeled as a shared finite state machine. Players are only able to activate transitions in the game's state machine when their role becomes active, but this information is generally kept hidden from the other players.

CHAPTER 6

Indices and tables

- genindex
- search

Index

W

werewolf:, 9