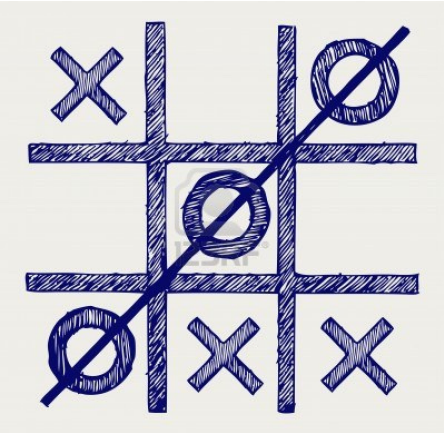
Tic-tac-toe slack custom command

Overview

This document presents the design of the tic-tac-toe ("TTT" below) Slack custom command. Ttt is a popular board game. The following wiki page has more details:
https://en.wikipedia.org/wiki/Tic-tac-toe



Slack is a cloud-based team collaboration tool. The Slack custom command manual can be found at:
https://api.slack.com/slash-commands

Essentially, the slack custom command is implemented by an http server that responds to POST or Get requests. This document discusses the http server design that is based on Python with Django web framework and SqlLite. The purpose of using Python and SqlLite is for fast developments and easy deployments, compared to other popular technologies, e.g. Java, mySql. The http service follows service-oriented MVC (Model-View-Controller) design.

Customer use case

Two users can play TTT in a channel, while other users in the same channel can watch. When a turn is taken that ends the game, the response indicates this along with who won. The user interface is available in the form of custom Slack command **/ttt**, following by the options below:

/ttt play-with @player --- start a new game with user. e.g. /ttt play-with @hliu
/ttt current --- The current ongoing game
/ttt place x y --- Place a piece at (x y). x y => [0, 3).
/ttt help --- Show this help page.

/ttt reset --- Stop the current game.

The game result can be:

- Win
- Draw

Requirements

1. The service should persist user data.
2. The service can recover from a crash, or new code deployment.
3. Users can create a new game in any Slack channel by challenging another user (using their @username).
4. A channel can have at most one game being played at a time.
5. Anyone in the channel can run a command to display the current board and list whose turn it is.
6. Users can specify their next move, which also publicly displays the board in the channel after the move with a reminder of whose turn it is.
7. Only the user whose turn it is can make the next move.
8. When a turn is taken that ends the game, the response indicates this along with who won.

Non-requirements

1. No concurrency considerations. Because of the nature of typing based communication, it is expected to see low chance of race in the same channel. The current design is immunized from race between channels.
2. Single node design. This design spec does not discuss the load-balancer, since single node should be able to handle the traffic at this moment.
3. No cache mechanism. We expect the traffic is relatively low in the first release.
4. Token verification.

Deliverables

1. A server that deploys the http service that implements the tic-tac-toe slack custom command
2. The source code of the http service in Python, available at:
   https://github.com/yonexer/slack_ttt
3. A design doc
4. A youtube video of a game set, https://youtu.be/WZfH7vnI1wE
5. A game in the general channel

Dependencies

We use Python 2.7 with Django web framework. SqlLite 3.0 is used for data persistence.

Detailed Description

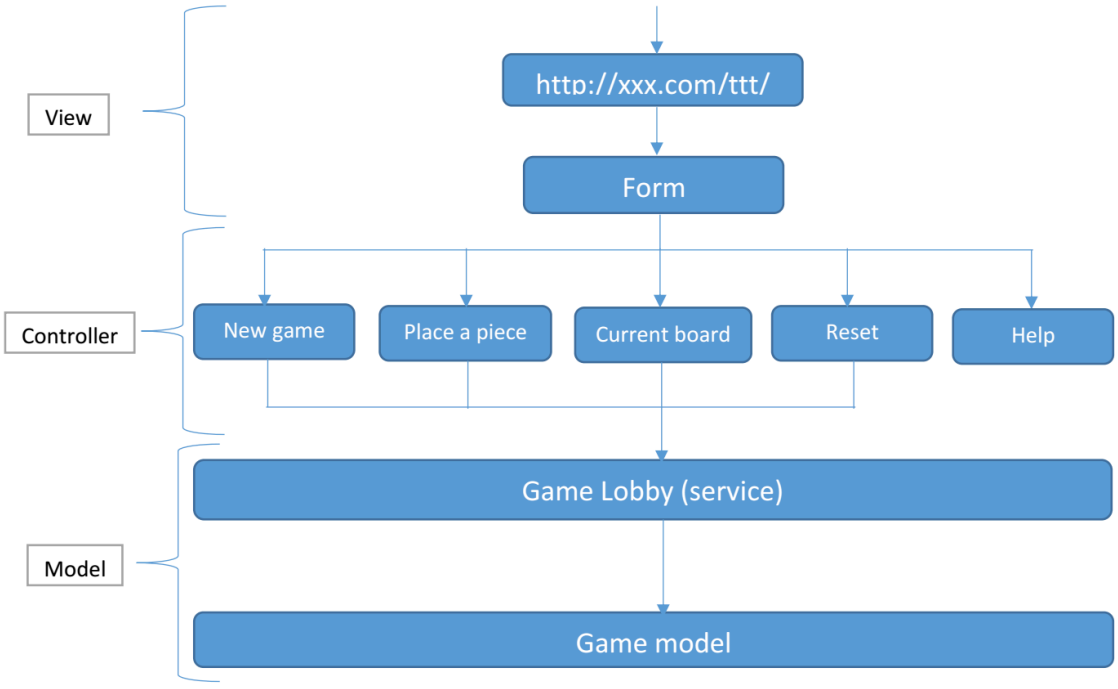
Slack custom command interface

As mentioned, the Slack custom command sends http requests to a URL. In this design, we use http://xxx.com/ttt/. Https is not supported for now. The parameters are sent in the Post body. Below is an example of the Post contents.

token=gIkuvaNzQIHg97ATvDxqgjtO
team_id=T0001
team_domain=example
channel_id=C2147483705
channel_name=test
user_id=U2147483697
user_name=Steve
command=/ttt
text=play_with @hliu
response_url=https://xxx/ttt/

Architecture

The http service follows service-oriented MVC (Model-View-Controller) design.



View layer

The view layer is a bit thin, due to the nature of Slack command. All input info is in a form, and all outputs are in plain string.

Controller layer

5 controllers are implemented, *start a new game*, *place a piece*, *show current board*, and *reset*. The controllers talk to the service layer to perform the functionality.

Model layer

Model layer consists of two parts. One is lobby, or game services, one is the data model of TTT game (more details below).

Data model

Each game is represented by:

- *channel_id*, the Slack channel that the game happens
- *player1*, the name of player1
- *player2*, the name of player2
- *last_player*, the name of the player who did the last move
- *board*, the current board status
- *game_id*

Token verification

In the first release, we don't bother with verify the request with the Slack service. The reasons are:

1. Low user count
2. Additional resources for verification
3. No credential user data

Testing Strategy

The Slack command is tested manually. And the data model is covered by unit tests. At this moment, we focus on happy cases.

Deployment

Please first install the dependencies. Then run the following commands:

```
cd $WORK_DIR
python manage.py migrate
python manage.py runserver 0.0.0.0:80
```

One game example

Please check:

https://youtu.be/WZfH7vnI1wE