Slack Tic-Tac-Toe

Objective: To implement Tic-Tac-Toe on Slack Chat as specified in https://slack-files.com/T024BE7LD-F0NJM55RV-cba031f54a

Author: Bharath Boregowda (bharath.boregowda@gmail.com)

This is the documentation of the Tic-Tac-Toe game implementation. The document describes the game features and the details of implementation.

Table of Contents:

1. How To Play:

Tic-Tac-Toe game can be invoked from the channel by using the command `/slack_ttt` or `/slack_ttt help`

Invoking the above command displays the list of available commands which are simple and comprehensive to start a game and take it all the way to the end.

```
Slack Tic-Tac-Toe:
1. /slack_ttt help - To display this message at any time
2. /slack_ttt status - To show current status of game in the channel.
3. /slack_ttt challenge @player_name - To start a game against @player_name.
4. /slack_ttt mark <1-9> - To mark a spot number between 1-9 during the game.
5. /slack_ttt abandon - To abandon current game. Only participants can abandon.
```

The game recognizes only the above commands as valid and any other commands will be ignored and the help message will be displayed.

As per the requirements, only one game can be in play in a given channel at any given time. Any user can start a game with another player by using `/slack_ttt challenge`

`@player_name`. A new game with fresh board will be created for the two players in the current channel, with the initiating player as the starting player of the game. The starting game board is displayed and the challenged user is notified as well.

```
| - | - | - |
|---+---+---|
| - | - | - |
|---+---+---|
| - | - | - |
```

The players can mark (`/slack_ttt mark <1-9>`) any spot on the board by addressing them using spot numbers from 1 to 9. If a player enters a number outside of this range or if he tries to mark a spot that is already marked then he is warned appropriately. If a player tries to play out of turn, he is warned to wait his until the other player completes his move. At the end of the turn, the game state is evaluated. If we have a winner or if the game ends in a draw, an appropriate message is displayed announcing the outcome of the game and the game is marked complete.

```
| 1 | 2 | 3 |
|---+---+---|
| 4 | 5 | 6 |
|---+---+---|
| 7 | 8 | 9 |
```

At any given point in the game, the participating players can abandon the game if a need arises. (`/slack_ttt abandon`). In such a case the game is marked as abandoned and the channel is ready to host a new game. Any member of the channel can invoke the help or status of the game at any point in time without affecting the state of a game if in progress.

2. Sample game states:

```
| O | X | O |
|---+---+---|
| - | X | X |
|---+---+---|
| - | X | O |

Game over! @Alice wins the game!
```

```
| O | X | O |
|---+---+---|
| O | X | X |
|---+---+---|
| X | O | X |

Game drawn!
```

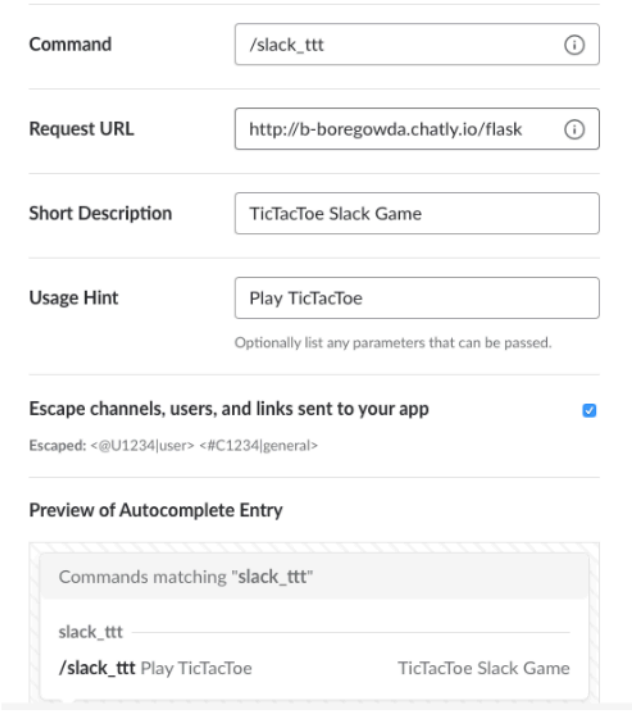
3. Implementation/Setup:

Now for the technical aspects of the game.

The backend for Slack slash TTT command processor is implemented in **Python** with **Flask** as the backend server. **MySQL** is used as the datastore for persisting necessary game states. The setup steps for each module are detailed below.

The Slack App:

The slack app is setup to send the /slack_ttt messages to the server at url http://chatly.io/flask



Flask setup:

1. The Flask app, named `slack_app.py` is located in the directory `/var/www/html/flaskapp`.
2. It is configured to listen on port 5000 and to process requests from open world, the internet. `app.run(host='0.0.0.0', port=5000, debug=False)`.

3. Additionally, Apache server is configured to reroute all traffic coming on port 80 with url /flask to the flask app which is listening on port 5000. `ProxyPass /flask http://127.0.0.1:5000/` is added in the file `/etc/apache2/apache2.conf`.

MySQL setup:

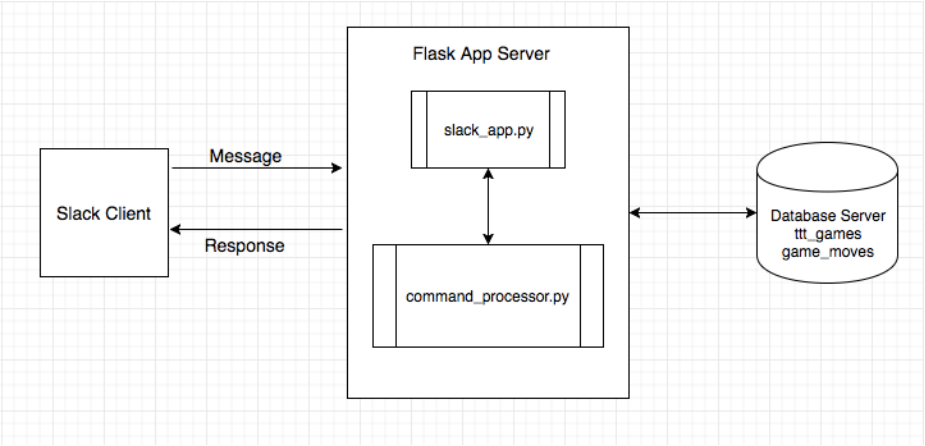
1. MySQL server is used as the database for persisting game data.
2. A database called 'slack' is created which will contain two tables 'ttt_games' and 'game_moves' (explained later).
3. A user 'slack_user' was added and granted permission to interact with the db tables `create user slack_user ; grant all on slack.* to 'slack_user'@'localhost' identified by 'slack_password';`
4. The table schema are in the file db_tables.sql and is checked-in along with the code files.

How To Run:

The game is currently running in screen called 'ttt' on the server provided.

```
FLASK_APP=/var/www/html/flaskapp/slack_app.py flask run --host='0.0.0.0' --port=5000 &
```

4. Design



The backend flask server is organized as follows:

```
Tic_Tac_Toe
      |- command_processor.py
      |- slack_app.py
```

```
|- config_helper.py
|- game_move.py
|- ttt_game.py
```

`slack_app.py` and `command_processor.py` handle the core game logic while `ttt_game.py` and `game_move.py` are the model files that handle everything related to game state persistence and retrieval.

The game states are organized into two tables, namely, `ttt_games` and `game_moves`.

```
ttt_games (
id INT,
channel_id INT,
player1_id STRING,
player2_id STRING,
current_player STRING,
status INT,
winner STRING,
date_created TIMESTAMP,
date_modified TIMESTAMP
)
```

ttt_games records every game that is created on any channel between every pair of players. The table has game id, the channel where it was initiated, player 1 who initiated the game, player 2 who he is playing against and also which player's

turn is to play in the current_player field. The status of a game can be `IN_PROGRESS`, `COMPLETED or ABANDONED`.

```
mysql> select * From ttt_games;
+----+------------+------------+------------+----------------+--------+-----------+---------------------+---------------------+
| id | channel_id | player1_id | player2_id | current_player | status | winner    | date_created        | date_modified       |
+----+------------+------------+------------+----------------+--------+-----------+---------------------+---------------------+
|  1 | C8947SKJQ  | U876W903Z  | U8GQG54ES  | U876W903Z      |      2 | U876W903Z | 2017-12-18 01:08:45 | 2017-12-18 05:51:21 |
|  2 | C8947SKJQ  | U876W903Z  | U8GQG54ES  | U876W903Z      |      2 | U876W903Z | 2017-12-18 06:04:49 | 2017-12-18 06:06:37 |
|  3 | C8947SKJQ  | U8GQG54ES  | U876W903Z  | U8GQG54ES      |      2 | U8GQG54ES | 2017-12-18 06:18:44 | 2017-12-18 06:20:27 |
|  4 | C8947SKJQ  | U876W903Z  | U8GQG54ES  | U8GQG54ES      |      2 | U8GQG54ES | 2017-12-18 06:51:57 | 2017-12-18 06:59:05 |
|  5 | C8947SKJQ  | U876W903Z  | U8GQG54ES  | U8GQG54ES      |      2 | U8GQG54ES | 2017-12-18 07:08:03 | 2017-12-18 07:11:00 |
|  6 | C8947SKJQ  | U8GQG54ES  | U876W903Z  | U8GQG54ES      |      2 | U8GQG54ES | 2017-12-18 07:11:28 | 2017-12-18 07:16:09 |
|  7 | C85ERP85N  | U8GQG54ES  | U876W903Z  | U8GQG54ES      |      2 | U8GQG54ES | 2017-12-18 07:13:40 | 2017-12-18 07:17:57 |
|  8 | C85ERP85N  | U876W903Z  | U8GQG54ES  | U876W903Z      |      2 | U876W903Z | 2017-12-18 07:52:48 | 2017-12-18 08:11:34 |
```

Game Moves

```
game_moves (
 id INT,
```

```
  game_id, INT,
  move_number INT,
  player_id STRING,
  game_board STRING,
  date_created TIMESTAMP
  )
```

game_moves records the moves a player makes in a particular game. game_id identifies the game and the move_number is the number of move so far in the game. A game will have at most 9 moves. game_board is the string representation of the game board which is an array of 9 spots.

```
mysql> select * From game_moves;
+----+---------+-------------+-----------+-----------------------------------------+---------------------+
| id | game_id | move_number | player_id | game_board                              | date_created        |
+----+---------+-------------+-----------+-----------------------------------------+---------------------+
|  1 |       1 |           1 | U876W903Z | ["-", "-", "X", "-", "-", "-", "-", "-", "-"] | 2017-12-18 05:32:57 |
|  2 |       1 |           2 | U8GQG54ES | ["-", "O", "X", "-", "-", "-", "-", "-", "-"] | 2017-12-18 05:38:38 |
|  3 |       1 |           3 | U876W903Z | ["-", "O", "X", "-", "-", "-", "-", "-", "X"] | 2017-12-18 05:40:33 |
|  4 |       1 |           4 | U8GQG54ES | ["-", "O", "X", "O", "-", "-", "-", "-", "X"] | 2017-12-18 05:41:41 |
|  6 |       1 |           5 | U876W903Z | ["-", "O", "X", "O", "-", "X", "-", "-", "X"] | 2017-12-18 05:50:42 |
|  7 |       2 |           1 | U876W903Z | ["-", "-", "X", "-", "-", "-", "-", "-", "-"] | 2017-12-18 06:05:02 |
|  8 |       2 |           2 | U8GQG54ES | ["-", "-", "X", "-", "O", "-", "-", "-", "-"] | 2017-12-18 06:05:27 |
|  9 |       2 |           3 | U876W903Z | ["-", "-", "X", "-", "O", "X", "-", "-", "-"] | 2017-12-18 06:05:44 |
| 10 |       2 |           4 | U8GQG54ES | ["O", "-", "X", "-", "O", "X", "-", "-", "-"] | 2017-12-18 06:06:29 |
| 11 |       2 |           5 | U876W903Z | ["O", "-", "X", "-", "O", "X", "-", "-", "X"] | 2017-12-18 06:06:37 |
| 12 |       3 |           1 | U8GQG54ES | ["X", "-", "-", "-", "-", "-", "-", "-", "-"] | 2017-12-18 06:19:12 |
| 13 |       3 |           2 | U876W903Z | ["X", "O", "-", "-", "-", "-", "-", "-", "-"] | 2017-12-18 06:19:21 |
| 14 |       3 |           3 | U8GQG54ES | ["X", "O", "-", "-", "X", "-", "-", "-", "-"] | 2017-12-18 06:19:28 |
| 15 |       3 |           4 | U876W903Z | ["X", "O", "O", "-", "X", "-", "-", "-", "-"] | 2017-12-18 06:19:36 |
| 16 |       3 |           5 | U8GQG54ES | ["X", "O", "O", "X", "X", "-", "-", "-", "-"] | 2017-12-18 06:19:46 |
| 17 |       3 |           6 | U876W903Z | ["X", "O", "O", "X", "X", "-", "O", "-", "-"] | 2017-12-18 06:20:19 |
| 18 |       3 |           7 | U8GQG54ES | ["X", "O", "O", "X", "X", "-", "O", "-", "X"] | 2017-12-18 06:20:27 |
```

1. `slack_app.py` receives the request from the slack client and forwards it to `command_processor.py` for interpreting the command sent in the message.
2. The `config_helper.py` handles connections to database server. Since this is a simple application with only one database and two tables, the config helper does not have much work to do. It would definitely become complicated once we have multiple database servers and connections to handle.
3. The command_processor first breaks down the message into words. It processes only the following five commands: `help`, `status`, `challenge`, `mark`, `abandon`. Any command other than these five will be ignored and help message will be displayed instead.
    a. `status`: This command retrieves the current status of game on the channel. If a game is in progress, the current game board and player's turn is displayed.
    b. `challenge @player`: This command takes a slack username as argument and initiated a new game with that user if there are no games currently in progress. A new blank game board is displayed and notifies the player's turn. If a game is in progress, a warning is displayed.
    c. `mark <1-9>`: This command takes a number from 1-9 and marks that spot with the player's symbol if that spot is not taken. If a player invokes this command out of

turn then a warning is displayed.

   i. After each valid move, the game state is evaluated to check if there is a winner. This is done by checking for consecutive presence of a player's symbol in a row or a column or a diagonal. If there is a winner, he is declared so and the game is marked COMPLETE with the winner as well recorded in the game state.

   ii. If however, the game has filled all the spots with no winner, then it is declared a draw and the game is marked COMPLETE with no winner for the game.

   iii. If the above two conditions are not met, then the game continues with the next player's turn.

d. `abandon`: If for some reason the game on the channel cannot be continued, either of the participants can abort the game by issuing this command. Only participants of the game in progress can abandon it and no other members of the channel can perform this operation.

e. `help`: This command displays the help message at any time of the game and can be invoked by playing and non-playing members of the channel.

f. Any other unrecognized command will be ignored and help message is displayed.

g. Command Processor makes use of a lot of helper function to extract user IDs from unfurled username mentions, to format username mentions, to pretty print the game state board, to evaluate the game winner and so on.

5. Future Scope

This is a simple implementation of Tic-Tac-Toe and clearly there is a lot of room for improvement in terms of design and scalability.

1. **Scalability**

   a. Flask is a robust server framework that can handle several concurrent requests and it fits perfectly for the current use case. However, if the number of concurrent users increase by magnitude then the flask app server would definitely see performance issues in handling multiple requests and responding within 3 seconds which is the cut off time for slack slash command messages.

   b. Writes to the database server are currently implemented without any transactions and retries because the work load for this implementation does not require it. If this app becomes write-heavy then we will have to invest in making db writes robust with retry and transactional support. If read traffic increases then we might have to bring up passive read database slave servers that could serve read requests. Caching could also help in this scenario.

   c. Game state evaluation and checking for winner is implemented in a naive manner for the simple fact that the game board id 3x3 size. This check could definitely be implemented with fewer comparisons.

2. **Code Structure**:
    a. Code structure could be rearranged, with more helper functions for formatting messages and game board displays.
    b. Currently, all game moves are recorded in game_moves database table which could be used for replay purposes. So every game at the most would need 9 moves or rows to be added in the table which could be considerable amount of data duplication. We could reduce this and maintain only the latest game state if storage space becomes an issue.
    c. Test cases for command processor and helper functions would benefit code quality.
3. **Use Cases**:
    a. Support for more commands/actions. As of now, a game can be abandoned only by the participating players. If both these players exit the channel/workspace with the game in progress condition then there is no way of aborting the game. Probably an admin could step in and abort the game in such scenarios.