DOCUMENTATION: MEMORY GAME APPLICATION

A CS 145 Requirement

ANDRES, Mireya Gen
QUIAZON, Niña Kamilla
RABINO, Andie

I. PROTOCOL

The following are the specifics of the protocol implemented in the Memory Game.

A. Message Format

The message is implemented as a dictionary, with the following required fields for the header:

a. Messages sent from client to server

i. `state` – informs the server of the current state of the client. This is used by the server to determine what action they should take.

ii. `game_state` - pertains to the current state of the game of the particular client. A client may have a different `game_state` from the other client.

b. Messages sent from server to client

i. `state` - sends `"OKAY"` if the server successfully processed the message.

ii. `game_state` - the `game_state` the client should transition to.

iii. `msg` - short description of the message (included for debugging purposes).

Messages may also contain other fields not listed above. While it is not mandatory to include these fields in every message, they are required to be sent during certain states of the game. For example, clients must send the index of the cards they flipped during the end of their `"PLAY"` state, but it is not required to send this information during the other states, such as `"WAIT"` and `"SETUP"`.

B. Actions taken by the Client and Server

a. The Server

The server determines what action to take based on the `state` sent by the Client. In simpler words, all backend computations and assignments are done by the server. Examples of `states` include:

`"CONNECT"` - indicates that connection has been established on the Client side; upon receiving this state, the Server will send data on how the game should be set up.

`"SETUP OKAY"` - means that the Client successfully set up the game (both Clients should have the same setup).

`"PLAY OKAY"` - means that a Client's turn is over; upon receiving this state, the Server will compute the Client's score and will send a message to the client to change their `game_state` to `"WAIT"`. Then it will send a message to the other client to change their `game_state` to `"PLAY"` as it is now the start of their turn.

`"END"` - means that all cards have been flipped and the game is over. The server will send messages to both clients to change their `game_state` to `"END"`. It will also send information about the player's scores to determine the winner of the game.

`"QUIT"` - means Client will disconnect from the Server; the Server will then notify the other player about this.

b. The Client (Player)

The client determines what action to take based on the `game_state` the server sends. In lighter terms, the Client is responsible for frontend implementation of the Server's messages and instructions. With this, the client will change their `game_state` to the one indicated by the message. Then, it will check for the optional fields depending on the `game_state`.

Examples of `game_states` include:

`"SETUP"` – means that the server has sent information regarding how the game should be setup. The Client will search for additional fields (such as `'index_list'`) required for setting up the game.

`"PLAY"` - means that it is the Client's turn to play the game. During this phase, the other player's `game_state` will be changed to `"WAIT"`.

`"WAIT"` - means that it is the other player's turn to play the game. The Client will wait for the server to send additional instructions.

`"END"` - means that the game is over. The `game_window` will now display the scores of the players and the winner of the game.

Note: the different `game_states` are stored as integer keys of a dictionary stored in `resources.py`.

II. APPLICATION

A. Instructions on how to build and run the application

The application can be downloaded from *github.com/ohandie/memorygame*. The final and working application can be found in the *version4* folder. Upon accessing said folder, run the `server.py` file on the terminal to run the server. Run the `client.py` file to connect to the server. Input the host, port, and desired username. The player may opt to have no username; their default username will be "Player 1" or "Player 2".

The game requires two players to connect to the server.

B. Minimum requirements and dependencies

In order to be able to use the application, one should have *Python 2.7.3* installed on his PC or laptop. The *pyglet module* will also be needed, as the application was built using pyglet.

Note that this application has only been tested on Linux machines (specifically, Ubuntu LTS 12.04). It is best to run the application on Unix machines.