Predict the Pic For Users

Available at www.predictpic.us

GAME Functionality

How to play

1. Input user name.
2. Click play
3. The image will slowly appears for the player to guess what it is.
    a. Player may need to wait for the round to start.
    b. There will be a question appear on the top of the image.
4. Player may guess the image by clicking on the letter at the bottom of the image.
    a. Click directly will relocate the button to the answer region
    b. Click on the letter at the answer region again to remove it, if misspell
5. Click ‘Confirm’ to check the answer
    a. The answer will not count if the time run out if the player did not click ‘Confirm’
6. Wait for the next image
7. The game will restart if no more life left. By clicking ‘Restart’

Game Mechanic

- Each round, one image, every player will start at the same time. Therefore if the player sign in at the middle of the round they will need to wait for the round to end.
- There will be 30 seconds time limit for each round to answer. There will be 5 seconds at the end of each round for the players to rest and prepare for the next image.
- Each players will have 3 lifes assigned to them. This life will only reduce when the round end and the player unable give a correct answer.
- The player can answer as many time as they want in each round with no consequences.
- The player can play the next round as long as they have at least one life left.
- The score will be based on how fast the player are able to answer the question correctly.
- The user must sign in again as a new player if they have no life left and wish to play again. All their previous progress can not be continue, but it will be saved.

Other

- At the sign in page there will be a list of top 10 highest score at the bottom of the page.
    - The list will contain their sign in name and their final score.
- In the game page, there is a list of all current player on the left side of the page. This will contain other user’s sign in name and their current score.

Predict the Pic For Developers

Frameworks, Modules, Libraries, etc:

- Semantic UI
- Node.js
- Express
- Socket.io
- Mongoose
- MongoDB
- Body parser
- Request

Installation

Set up

1. Download the app source code and it extract to a directory.
https://github.com/RodrigoFigueroaM/GuessThePic.git
2. Install MongoDB
3. Install foreman.

Running the app

1. Type in Terminal: npm install to install all of the dependencies
$ npm install

2. Run the service
$ foreman start
foreman run:

- Mongo DB
- node server.js
- node socketIO.js

3. Access the website
Go to browser and type localhost:3000

File Breakdown

Folder: **GuessThePic**

- server.js

    - Establishes routes to handle database requests.
- SocketIO.js
    - Establishes routes for websocket. Primarily used as an intermediary to communicate with clients and server.

Folder :GuessThePic/public/

- addInfo.html
    - Html file that allow input information to database
- help.html
    - Html file with additional information on how to play the game
- index.html
    - Html file with main html body for game

Folder: GuessThePic/public/css

- sytle.css
    - File with css to manipulate game looks

Folder: GuessThePic/public/js

- app.js
    - Used for communication with server to add information to database
- clientSocket.js
    - Used to communicate with socket throughout the game progress
- imgDisplayer.js
    - Used to manipulate canvas and display picture for game
- IndexUpdate.js
    - Used to retrieve top ten scores from server
- inputCheck.js
    - Used to create dynamic buttons from the socket request.
    - Used to check the user input.
- knockOutControllers.js
    - Knockout integration. It contains all controllers, functions and classes used with knockout bindings.

Folder: GuessThePic/public/Semantic/dist

- import from Semantic UI

Maintenance

- There are ../addInfo.html page use to add in the question and score to the server directly.

- There are slot for for each component needed for the question, include question, answer and image's URL.
- Player name and score also can be added to pre made the score board.

** The question need to be added first before the game will run.

Server

The server use MongoDB to store two table.

- QuestionDB - contain question, answer, id and image's URL.
- ScoreDB - contain username and score.

All answer will be converted to Uppercase for uniformity and easier to compare.
The photo in the API may be to specific or vague for the scope of the game.

Server response

(Everything String In server will be in uppercase except url)

Retrieve random picture and answer
POST :/question
IN : {catagory: 'city'}
RETURE : {questionOut: 'what is this', answer:'book', pic:'<url>'}

Send in the finish score
POST: /scoreUpdate
IN :{userName:'someone', score:1000}
RETURN :Success

Retrieve top 10 score
GET :/score
RETURE : { 'scoreLenght': <numberUser>,
'scoreSort':[
{UserName:<someone>, UserScore:<1000>}
] }

Socket

All interaction to the web will be through the socket.io. Socket.io will be the only entity that access the server directly, excluding ./addInfo.html. All score and answer calculation are performed in the socket.io server. The timer also handle by the socket server. At the end of the player game life the total score will be send to the server to store.

Socket server will request the answer from server and distribute it to every player at the same time. The socket server will keep on distributing the questions until no more players playing.

Socket IO function

- new user
  IN: {username: <username>}
  OuT “get user”: [{userId: <id>, username: <name>, life: 3, score: 0}]

- Question send every 35 sec
  IN: NONE
  OuT “get question”: {picture: <url>, question: <question>, timer: <30>}

- End round
  In: NONE
  Out: if life > 0:
  send client “check userLife” : {userLife: <num>}
  else
  Send server client score
- play again
  In: NONE
  Out: {waitTime: <millisecond>}

- answer
  IN: { answer: <answer>}
  OuT: {result: <true or false>}

- score
  IN: NONE
  OuT:
  ```
  {
  score: [
      {
          username: <name>,
          userScore: <score>
      },
      {<more users score>}
      ]
  }
  ```

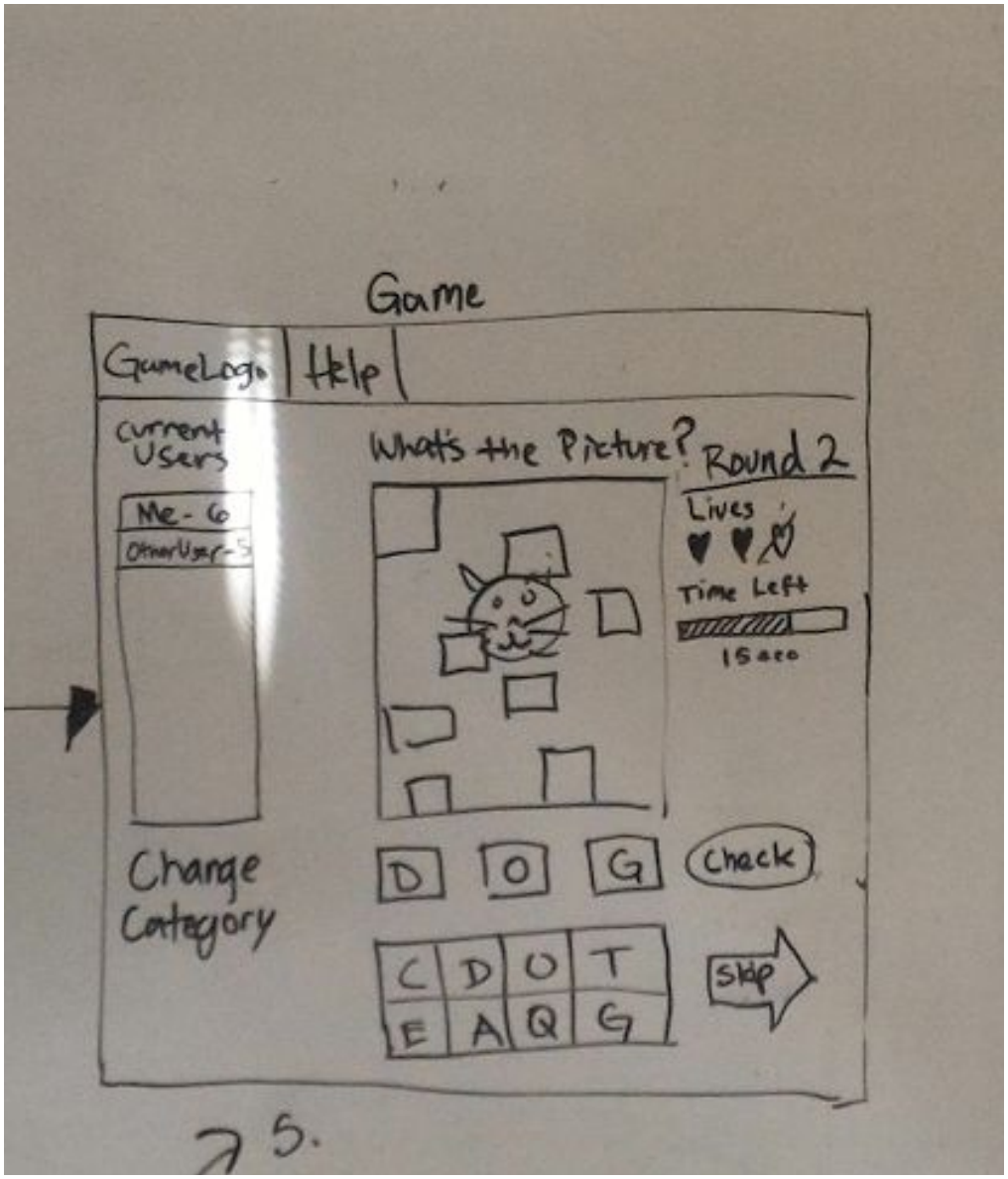
Development History

ToDo:

UI

- Decide colors
- Logo
- Letters (working on this)
- Buttons
    - Check
    - Skip
    - Lives

And I wanted to dump my thoughts out more on the logic



So we will have blank boxes on top (answer) and the letters to choose from below

1) To get the number of boxes for the answer, we need to count the number of letters. This will be dynamic as not all answers are the same length.
    a) How will this be passed? JSON object like {answer: cat} ? Do we need an array?
    Reply: no need for the array for answers because when they finish 1 picture JS (socket IO) will auto request for the next picture so the socket io can send both picture (not sure how to pass image yet) and answer (json) to all users
2) For the letters to choose from:
    a) We need the letters that make up the answer PLUS how many more?
    b) Will there always be the same number of letters to choose from?
        i) We can have set of array with 5 character and randomly added to the

Answer string then show it.

c) Randomize the placement. (Maybe have a scramble button?)

3) Once letters are dragged to answer, how do we know what is correct and incorrect? By the name of the image file? (save the letter a as “a.jpg”, b as “b.jpg”, etc…)

PROBLEM: I could not find a reliable Free API for photo and additional keyword must be added to the database therefore it might be better to store the url directly

What function do you need from the server?

Server response (Everything String In server will be in uppercase except url)

Retrieve random picture and answer
POST :/question
IN : {catagory: ‘city’}
RETURE : {questionOut: 'what is this', answer:'book', pic:'<url>'}

Send in the finish score
POST: /scoreUpdate
IN :{userName:’someone’, score:1000}
RETURN :Success

Retrieve top 10 score
GET :/score
RETURE : {
'scoreLenght': <numberUser>,
'scoreSort':[
{UserName:<someone>, UserScore:<1000>}
]
}

Socket

new user
IN: {username: <username>}
OuT “get user”: [{userId: <id>, username: <name>, life: 3, score: 0}]

Question send every 35 sec
IN: NONE
OuT “get question”: {picture: <url>, question: <question>, timer: <30>}

end round
In: NONE
Out: if life > 0:
send client “check userLife” : {userLife: <num>}

else

Send server client score

play again

In: NONE

Out: {waitTime: <millisecond>}

answer

IN: { answer: <answer>}

OuT: {result: <true or false>}

score

IN: NONE

OuT: {

score: [

{

username: <name>,

userScore: <score>

},

{<more users score>}

]

}