Maintenance guide

In this section we will explain what are the requirements of software and hardware

that must exist in order to use \ continue to develop the system.

1. **Hardware and Software requirements**
   1. **Used during the development**

* Google Colab -enviroment
* Python -main developing programming language
* JavaScript -for manipulate and control the HTML
* HTML – display and GUI of the game
* Jinja - for HTML and JavaScript python templets
* CSS- for styling
* Firebase library – database
  1. **Source code and python environment**

Required:

* Google account
* Google Drive access
* Wi-Fi access
* To run the game, first run the last cell (“CSS”), and the click on run on the tool bar-> click run all
  1. **Database environment**

The Database maintained on firebase and in order to get access it needed invitation by email.

1. **Important files:**

Code: pythongame.ipynb

1. **Main functions:**
   1. **login:**

Receives a username and password and checks if the username exists and assigns it the appropriate screen (if it's an administrator, open the administrator screen, if it's a user, open the game screen).

def login(username, password):

* 1. **loadQuestions:**

A function that returns an array of questions object, the function extracts the questions from the database and for each question creates a question and inserts it into the array.

def loadQuestions():

* 1. **run\_end\_game:**

A central function in the code that displays the game end screen.

def run\_end\_game():

The `run\_end\_game()` function is part of a larger code snippet. the `run\_end\_game()` the function performs the following actions:

1. It attempts to retrieve the `username` using the `user.get\_username()` function. If an exception occurs during this process, the `temp` variable is set to `None`.
2. If `temp` is not `None`, it imports the necessary modules (`IPython` and `output` from `google.colab`).
3. It attempts to retrieve the `username` again using the `user.get\_username()` function. If an exception occurs, it prints a message stating that the user has not logged in yet and sets `tempUsername` to `None`.
4. It defines a nested function called `get\_user\_index(user\_name)` that iterates over a list of users obtained from the Facebook API and returns the index of a user that matches the given `user\_name`.
5. It retrieves a list of users and their scores from the Facebook API.
6. It defines two functions, `get\_score()` and `get\_questions()`, which appear to retrieve the global variables `score` and `questions`, respectively.
7. It calls the `get\_questions()` function to retrieve the questions.
8. It defines a function called `get\_user\_data(tempUsername, FBconn)` that uses the `get\_user\_index()` function to retrieve the user's index based on the `tempUsername`. It then retrieves the user's history from the Facebook API and stores it in the `list\_of\_history` dictionary.
9. If `tempUsername` is not `None`, it calls the `get\_user\_data()` function to populate the `list\_of\_history` dictionary.
10. It defines a function called `generate\_leaderboard\_and\_history\_html(tempUsername, FBconn, sorted\_list\_of\_scores, questions)` that generates HTML code for displaying the leaderboard and the user's history table.
11. If `tempUsername` is not `None`, it calls the `generate\_leaderboard\_and\_history\_html()` function to populate the `history\_html` and `leaderboard\_html` variables.
12. It defines a string variable `html\_end\_game` that contains HTML, CSS, and JavaScript code for displaying the end game screen. This code includes a leaderboard table, a history table, tabs for switching between different views, and buttons for playing again or returning to the main screen.
13. It defines a function called `mainScreen()` that clears the output and displays the main screen using IPython.
14. It registers callbacks for the "handle\_return\_to\_main\_screen" and "handle\_quiz" events to handle button clicks in the HTML code.
15. It displays the end game screen by calling `IPython.display.HTML(html\_end\_game) `.

The `run\_end\_game()` function does not explicitly return any values. Instead, it executes the code to display the end game screen.

* 1. **Get\_firebase\_data:**

A function that receives an index of a question and retrieves the question from the database.

def get\_firebase\_data(selected\_index):

* 1. **insert\_new\_data:**

A function that receives a question and updates it in the database.

def insert\_new\_data(question):

* 1. **OpenQuiz:**

def openQuiz():

This is the main function of the game, which launches the user into the questions area. This function is called when the user chooses to launch a new game, and calls ‘displayQuestion’, which shows the user different questions. This function is called when the user clicks on start a new game.

* 1. **OpenRanking:**

def open\_ranking():

This function loads the user data from the database and shows his ranking compared to the rest of our users. This function is called when the ‘my rankings’ button is pressed at the home screen.

* 1. **displayQuestion:**

def displayQuestion(question\_num, question, score):

A function that modifies the HTML code to display the current question, and the rest of the variables of the current screen (quiz mode).

* 1. **handleAnswer:**  
     def handleAnswer(isCorrect,answer):

A function to handle the user’s input. If the user clicks on one of the answers, this function is called, checks input for correctness, updates the game variables and advances the user to the next question (or, if final question has been reached, to the end-game screen).

* 1. **endGame:**

def endGame():

A function to load the next colab code-block’s end-game code. Merely clears output and calls the run\_end\_game function previously mentioned.

* 1. **Insert\_graph\_data\_to\_db:**

    def insert\_graph\_data\_to\_db():

A function to insert session data (graph and score for a logged in user) to our firebase database at the end of the game.

1. **Interesting code snippets:**

During our journey, we found some interesting code snippets online and with each other, among which:

* 1. **Ranking**

html\_ranking = f'''

    <!DOCTYPE html>

    <html>

    <head>

        <title>Ranking</title>

        <style>

            .tab {{

                {tabCSS}

            }}

            .tabcontent {{

                {tabContentCSS}

            }}

            .tab:hover {{

                {tabActiveCSS}

            }}

            .tab-container {{

                {tab\_containerCSS}

            }}

            @keyframes fadeEffect {{

                {fadeEffectCSS}

            }}

            .container {{

                {container}

            }}

...

'''

This is an example of injecting CSS directly into the page using python variables, this technique speeds up the code in our colab notebook. This example also shows Python 3.6’s new f-strings which provide an easier way to control textual data in the language.

* 1. **Number of mistakes from Question Class**

//...

const mistakes = {[(question.mistakes or 0) for question in questions]};

//...

This example shows a neat Python trick: If a variable could be set to “null” (None in Python), and we want to take care of this edge case, we can use an ‘or’ conditional in our assignment, which will replace the ‘None’ variable with our wanted default value (0, in this case).

* 1. **Using a General pop up layout to insert it into the pages**

html\_popup='''<div id="popup-container" class="popup-container">

    <div class="popup">

      <h2>Help</h2>

      <hr style="width:50%;text-align:left;margin-left:0" >{{ content }}<hr>

      <button class="button close-button" onclick="closePopup()">Close</button>

    </div>

  </div>'''

script\_popup = '''function showPopup() {

      var popupContainer = document.getElementById('popup-container');

      popupContainer.style.display = 'block';

    }

    function closePopup() {

      var popupContainer = document.getElementById('popup-container');

      popupContainer.style.display = 'none';

    }'''

def insert\_popup\_content(pagetype):

  html\_popup\_with\_content=""

  match pagetype:

    case "home":

      html\_popup\_with\_content= html\_popup.replace('{{ content }}', '''<p>Hi, welcome to pythonGame, the trivia game that will teach you Python in the best way!</p>

      <p>There is 10 questions in a game</p>

      <p style="color:green; font-weight: bold;"> Correct answer you will get you +3 🪙 </p>

      <p style="color:DarkOrange; font-weight: bold;"> Each worng answer will cost you 1 🪙</p>

      <p>All you need to do is to answer the right question and you are ready!</p>

      <p>So Lets go we are waiting for you!.</p>''')

    case "quiz":

      html\_popup\_with\_content= html\_popup.replace('{{ content }}', '''

      <p>There is 10 questions in a game</p>

      <p style="color:green; font-weight: bold;"> Correct answer you will get you +3 🪙 </p>

      <p style="color:DarkOrange; font-weight: bold;"> Each worng answer will cost you 1 🪙</p>

      <p>on the next question button you continue to the next question or can skip a question</p>

      <p>on the previuos question button you can preview your answers or answer a skipped question</p>''')

    case "manager":

      html\_popup\_with\_content= html\_popup.replace('{{ content }}', '''

      <p>Here you can edit or add new questions</p>

      <p>to edit choose a question from the combo box and edit the field</p>

      <p>to add a new question please insert all of the question input field</p>

      <p>Remember to click on save :)</p>

      ''')

  return html\_popup\_with\_content

%store  html\_popup

%store  script\_popup

Include the content by calling the insert\_popup\_content method and then replace the variables In the screen display with the relevant content for the “?” button on the top left of the screen.

* 1. Include HTML cell content in other cell

In order to include it it needed to use the store command:

%store css

And in the dest cell it needed to include the %store -r command:

%store -r css

It allowed us to split the code into cells and maintain a logic order during the development.

1. **Design patterns:**

In our project, we used a singleton object in our database connector object. This means that a single connection to the database exists during the lifespan of our game (on the user’s Google colab cloud instance) which communicates with our firebase database, such that we do not waste time re-connecting or re-initializing this important controller.

initializing this important controller.