2019

A Game of Languages

FINAL PROJECT 9750

JULIEN RAPPE

Binding of Casper: A Game of languages

1 CONTENTS

2 INTRODUCTION

This paper was written as a final project for the "Software Tools" class at Baruch College and aims to help any enthusiast learn how to make their own JavaScript game available online. This is not the full project, but a short introduction for beginners explaining the basics of the languages needed in the project. By the end of the paper you should be able to understand how to make your own simple website and the structure of the full project available at https://github.com/rjulien1994/BindingOfCasper . First, we will cover some basic definitions. Then, we will go into some details for each language in the project. Finally, I will summarize my experience making "Binding of Casper" and the lessons I have drawn.

3 THE BASICS

First, let us understand how a website works. Imagine a restaurant where the kitchen is our server and the tables are the client's web browser. When clients arrive at their table, they can send an order to the kitchen. This is equivalent to typing the link of the website in the browser. Once the order is received, the kitchen gets the ingredients and prepares the meal. Upon completion, the kitchen sends it back to the table so that the client only sees the final product. In our case, the server collects and organizes the data in a way that is convenient for the client. Finally, we must give the necessary tools for the client to enjoy his food. Making tools available to clients without having to send additional requests to the server is the interactive part of our website.

The frontend languages used for the game are HTML, CSS and JavaScript. The HTML is the structure and layout of our webpage; it is where we will define all the elements that will be visible to the user. The CSS is the static design of the page; it will take the available elements in our html code and change its format to the one specified. The JavaScript is the dynamic content of the page; it can change the structure and style of our page dynamically without having to load external data.

The backend languages used are Python and SQL. Python is an interpreted, object oriented, high-level programming language. It is highly efficient to treat and handle data which made it popular among web developers. SQL is a special language, as it is command based and not scripting based. This means we will include SQL commands lines in our python script to interact with a database. However, there will be no SQL script running on our server.

Now that we defined the pieces of the puzzle, let's understand how we put them together. Since we are making a webpage, we will start with our web structure, the HTML. Once the structure is in place, we will style it with CSS and JavaScript. Finally, we will see how to host a website with Python. We will touch on communication with an SQL database but unfortunately, will not be able to go into the details.

4 MAKING A STATIC WEBPAGE

The static page is the skeleton of a website. It is the code the browser can interpret and print out in a manner that is readable by humans. As mentioned earlier, it is a combination of HTML and CSS that will load and organize the data on the page. It is worth noting that without an HTML file, we cannot run any JavaScript. In reality, one HTML file is enough to make a website, although it will probably be a boring one.

4.1 HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Tab Title</title>
  </head>

  <body>
    <h1>First Webpage</h1>
    <p>Hello World!!!</p>
  </body>
</html>
```

Figure 1

HTML consists of a series of elements defined by tags. Each tag specifies to the browser how to treat its enclosing. Each tag has a specific purpose and can take parameters such as “id” or “name”.

By simply typing the code in Fig. 1 in your text editor and saving it as name.html, you made your first website! Now, go in your browser and type the location of the html in your browser to see the result.

In “Binding of Casper” the HTML file is the bridge between the python code and the JavaScript. When the URL is requested, the HTML code is sent from the python code with all the required images and the current version of the game. During the game, the JavaScript creates a form in the HTML codes which is filled with the player’s in-game statistics. Once the game finishes, the HTML form is sent back to the server to be stored.

4.2 CSS

In “The Basics”, we have seen that HTML loads and creates elements on the page but does not take care of the appearance. At the same time, we do not want all the loaded images to show at once. Since this is a game they should only be displayed when needed. This is where CSS comes in. CSS can be added in a style tag in the HTML file. But it is always better to create a separate script and link it in the following way:

```
<link rel="stylesheet" type="text/css" href="style.css" />
```

Figure 2

In Fig. 2 the style.css would be the named of our script. It does not matter if you include your CSS directly in the HTML or in a separate file. The syntax stays the same.

```
img {
  display: none;
}

strong {
    font-weight: normal;
    font-size: 16px;
}
```

Figure 3

First, you declare tag you wish to style. Then, enter the parameters within curly brackets. Fig. 3 shows how to hide images using the “img” tag and change the size and weight of all texts enclosed in the “strong” tags.

If you have multiple tags of the same type but only want to change the style of one, you can give it an “id” and then reference it in the CSS file using # as follows:

```
#touchCommand {
    display: none;
}
```

Figure 4

You will see in my code that I have very little CSS as I mostly used the language to hide elements from the user. I then used the JavaScript code to display information based on the current user’s authorizations.

5 ADDING INTERACTIVE CONTENT

At this point, our website loads all the images and creates all the elements of the game without displaying them. This is where the fun begins. JavaScript can listen for events, change CSS and HTML elements and perform advance mathematical models which is perfect for a mini game.

5.1 JAVASCRIPT

Like CSS, JavaScript can be included in the HTML code within a "script" tag or in an external file. Save it with the extension .js and then link it as follows:

```
<script type='text/javascript' src="main.js"></script>
```

Figure 5

JavaScript is object-oriented and allows you to easily declare variables and change their types.

```
var x = 2;
x = 'hello'
let y = ' 2 ';
let sentence = {end: 'you'};

console.log(x+y+sentence.end);
```

Figure 6

As shown in Fig. 6, to declare a variable, you must start with the statement "var" or "let" regardless of their types. Variables can be numeric, string, dictionary…etc. Operators work just as well with number as strings. JavaScript allows you to declare functions with input parameters in order to perform actions or return values. You can also create *classes*, which are elements with functions and variables in them. Finally, each JavaScript line of code must end with a semi-colon.

When creating a game, you want to break its components down into classes. The classes will behave according to their own set of rules. This helps to structure to the game. For example, all the functions and parameters related to the player will be stored in its own class. Once a class is declared, it still needs to be called and stored in a variable to be used.

Here are some of the classes in my code:

- Game: Combination of all classes making the game
- Player: Handles everything in relation with player icon and stats
- KeyBoardHandler: Listens for events on keyboard or phone to control player
- Tears: Handles tears shot by player
- Enemies: Creates enemies according to the level
- GameMode: Takes care of in-game screen selection

As the game advances, JavaScript also stores the player's statistics each level in order to send it back to the server. This will be useful for development as it will allow us to see if any level is uncharacteristically difficult. It will also aid in finding out if the device on which the game is played has an influence on difficulty.

In order to interact with the HTML, we need the object document which can be interpreted as the visible webpage. The example fetches the canvas and stores the context in a variable in order to change its content later.

```
var canvas = document.getElementById("gameScreen");
var context = canvas.getContext("2d");
```

Figure 7

6 HOSTING YOUR SITE AT HOME

With all the pieces above, I created a game which I could access on my local machine's browser but nowhere else. Furthermore, since we have no servers, we have no platform to store data either. Luckily in this situation, Python has a very convenient package called Flask. Flask is a micro-frame platform which create a virtual server listening to the specified links and sends HTML responses. Unfortunately, Flask does not connect with any external SQL database. This is why I am also using "mysql.connector" which allows me to send command lines to my database to fetch, edit, and remove data.

6.1 PYTHON

Python flask allows anyone to make a web page using the following code.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run()
```

Figure 8

First, we load the Flask package to our script and create an instance of the server in the variable app. Once we declare the server, we can add routes. Routes are the requests the server will be able to answer to with the function right under it. Note that the "hello_world" function does not return its output to a console, but rather sends it to the web browser which submitted the request. Finally, the last two lines will run the server if the script is executed on your local machine.

Flask has many useful functions which were crucial to the well-functioning of my game. For example:

- Render_templates: Allows flask to load HTML and CSS files from machine
- Request: Allows Python to handle forms sent from HTML
- Flask_assets: Compiles CSS and JavaScript code to be readable by browser

Binding of Casper is made of 3 routes. There is the default page, the login page, and the game page. The default page redirects the user to the login page which then allows access to the game page. When the game loads, the past user's data are sent to the website with the purpose of personalizing their experience. The script is also in communication with an SQL database, sending information about the game every time a user ends a session.

7 BINDING OF CASPER: MY EXPERIENCE WITH THE GAME OF LANGUAGES

Using all the pieces discussed above, I developed a mini game called Binding of Casper. It is based on the arcade game *Binding of Isaac* and was my first test with JavaScript. You can play the 3rd beta of the game at bindingofcasper.xyz/ which will be online until December 22nd, 2019. This version requires you to sign up before playing on a computer or smartphone. With each new beta came new user feedback, which forced me to adjust how I tackled the project.

When working on the first version, I focused on collecting in-game data and storing it in a SQL database. I broke my database down into 3 tables: Players, Games and Levels. I was hoping to use this data to draw conclusions on each level in order to improve gameplay experience. However, when sharing the first beta, many users found it difficult to log in. Additionally, a portion of the data stored in the game table was corrupted.

This shifted my attention to the design of the page and the clarity with which to display information. Instead of using the quantile data in my database, I centered the development of the second beta on user feedback. But again, after releasing the 2nd beta, I did not collect enough information as many users unsuccessfully tried to use their phones to play.

Thus, for the last beta I worked on making the game available on smartphones and giving admins the ability to change their own stats. The goal was to try to appeal to more people. However, like the first beta, the feedback is that the touch commands are difficult to use.

Every beta has forced me to improve the design and code for the game. With each update, I learned new ways of writing code and functions in all 5 languages. However, the biggest lesson I learned is the importance of staying structured both in the code and display. Complicated code is pointless if it cannot be used with ease by the players. Furthermore, when I started making this game, I expected to improve each version using statistical analysis. But in truth, the most useful tool is user feedback because only an external point of view can detect the issues you miss.