



**UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS**

WIZARD GAME

ESTUDIANTES:

JOSUÉ RODRÍGUEZ - 20191020109
DAVID RAVELO- 20191020061

PROFESOR:

ALEJANDRO DAZA

Universidad Distrital Francisco José de Caldas
Facultad de Ingeniería
Ingeniería de Sistemas
Bogotá D.C.
2021

ÍNDICE GENERAL

1. Introducción	3
2. Descripción	3
3. Objetivos	3
3.1. Objetivo general	3
3.2. Objetivos específicos	3
4. Alcances y Límites	3
5. Tecnologías	4
6. Desarrollo	4
6.1. Funcionamiento Didáctico.	4
6.2. Funcionamiento Técnico.	6
6.2. Estructura de módulos y/o paquetes del proyecto:	8
6.3. Paradigmas de programación	9
6.3.1. Imperativo:	9
6.3.2. Orientado a objetos:	9
6.3.3. Scripting:	9
6.3.4. Funcional - Declarativo:	9
6.3.5. Back-end:	9
7. Conclusiones	10

1. Introducción

El proyecto hará uso del paradigma imperativo, el paradigma funcional, el orientado a objetos y el scripting vistos en el curso de modelos de programación II para la creación de un juego. Contendrá uso de bases de datos, tecnologías web de python como flask y la programación lógica.

Este programa se podrá encontrar en el siguiente repositorio:
https://github.com/davidravelo1/Wizard_Game.git

2. Descripción

El juego consistirá en mover un mago de arriba hacia abajo controlado por el usuario. Se tiene como objetivo esquivar cada uno de los proyectiles lanzados por la computadora y de esta manera ir acumulando puntaje que se obtendrá a partir del tiempo que dure (en segundos) sin ser impactado por dichos proyectiles.

Los proyectiles y obstáculos son generados desde el lateral derecho del mapa apuntando al mago, el cual estará ubicado en el lateral izquierdo. Además de esto, el puntaje se podrá ver en la esquina superior derecha.

El usuario podrá guardar su puntaje obtenido bajo un alias, para poder ser listado en un ranking de los mejores puntajes obtenidos globalmente que se irá actualizando a medida de que los jugadores rompan los puntajes máximos previos.

3. Objetivos

3.1. Objetivo general

- Crear un juego en un entorno web con temática de hechicería usando html, css, javascript, Flask, javascript y Mysql.

3.2. Objetivos específicos

- Implementar los paradigmas de programación vistos en clase, para el desarrollo del juego.
- Crear una base de datos que permita persistir el mayor puntaje y el número de intentos realizados por el jugador.

4. Alcances y Límites

- La primera versión del juego va a constar de láseres como enemigo, que irán aumentando su velocidad a medida que el puntaje aumente.
- El juego permitirá guardar el puntaje acumulado de cada jugador, de manera que se tenga una lista con los puntajes ordenados de manera descendente.

- Se buscará añadir complejidad al juego, mediante la aparición de más lasers, y enemigos que provoquen que la jugabilidad sea más compleja.
- En primera instancia Wizard game estará disponible de manera local.

5. Tecnologías

Las tecnologías empleadas para el desarrollo de Wizard Game son las siguientes:

- **Python:** Lenguaje de programación de alto nivel con el que se implementará parte del backend. Con python se establecerá la conexión a la base de datos.
- **Flask:** Librería de python que otorga un ambiente web, sobre el cual se desarrollará el backend y el frontend.
- **MySQL WorkBench:** Base de datos relacional en donde se persistirán los puntajes de los jugadores.
- **Javascript:** Lenguaje de programación con el cual se crean los scripts y funcionalidades del juego.
- **HTML5:** Lenguaje de etiquetas de hipertexto que permite estructurar el cuerpo de la página de presentación de juego, así como los títulos y textos que se encuentren en él.
- **CSS3:** Lenguaje de diseño que complementa al html5, otorgando estética y estilos.

6. Desarrollo

6.1. Funcionamiento Didáctico.

Wizard Game, en su página principal posee un menú de inicio con tres botones, los cuales permiten redirigir hacia las instrucciones del juego, hacia el ranking y hacia el juego en sí.



Imagen [1] Home.

Una vez hecho click sobre él botón de instrucciones, se obtiene la página en donde se describen las teclas de movimiento para interactuar con el personaje principal, “el mago”.



Imagen [2] Instrucciones.

De otro modo, cuando se realiza un click sobre el botón iniciar, se despliega la página del juego. El estado inicial del juego es en “PAUSE”, de modo que para iniciar el juego se debe oprimir la tecla “p”.

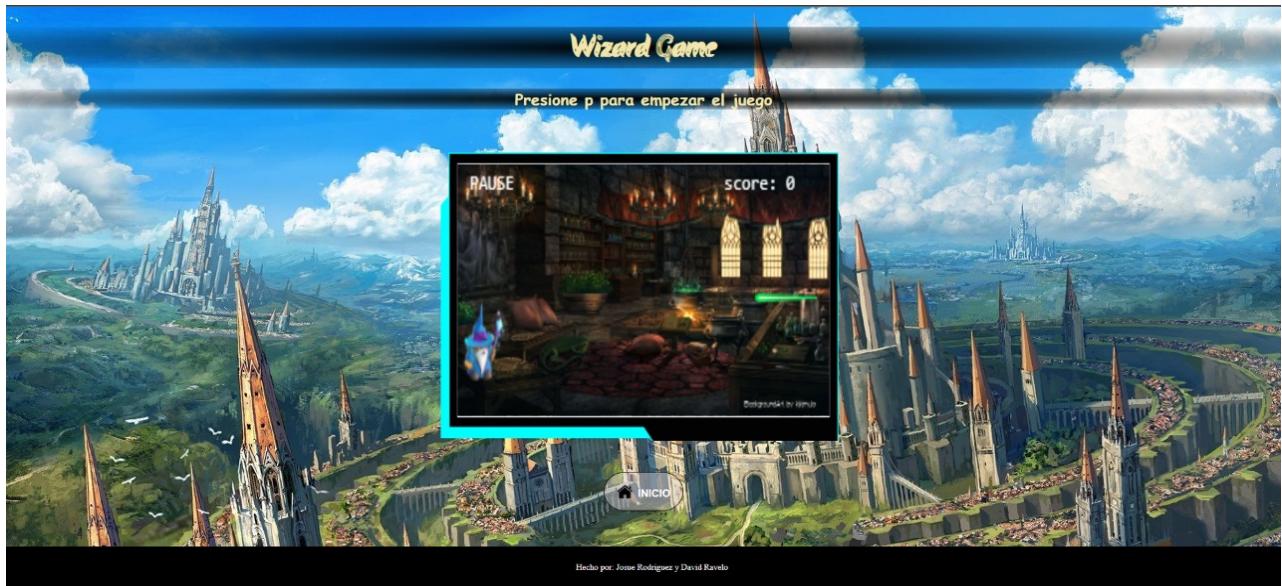


Imagen [3] Juego.

Una vez el juego esté corriendo, se podrá mover al mago con las teclas “w” y “S” tal como se describe en la página de instrucciones. Se debe tener cuidado con los movimientos ya que se deben evitar las colisiones con los láseres.

De chocar con algún láser, el tablero del juego se deshabilitará y en su lugar, se mostrará un formulario para ingresar el alias bajo el cual se quiere guardar el puntaje obtenido.



Imagen [4] Puntuación.

Una vez se guarda el puntaje alcanzado en Wizard Game, este podrá ser consultado en la página de ranking, mediante una tabla que está ordenada de manera descendente.

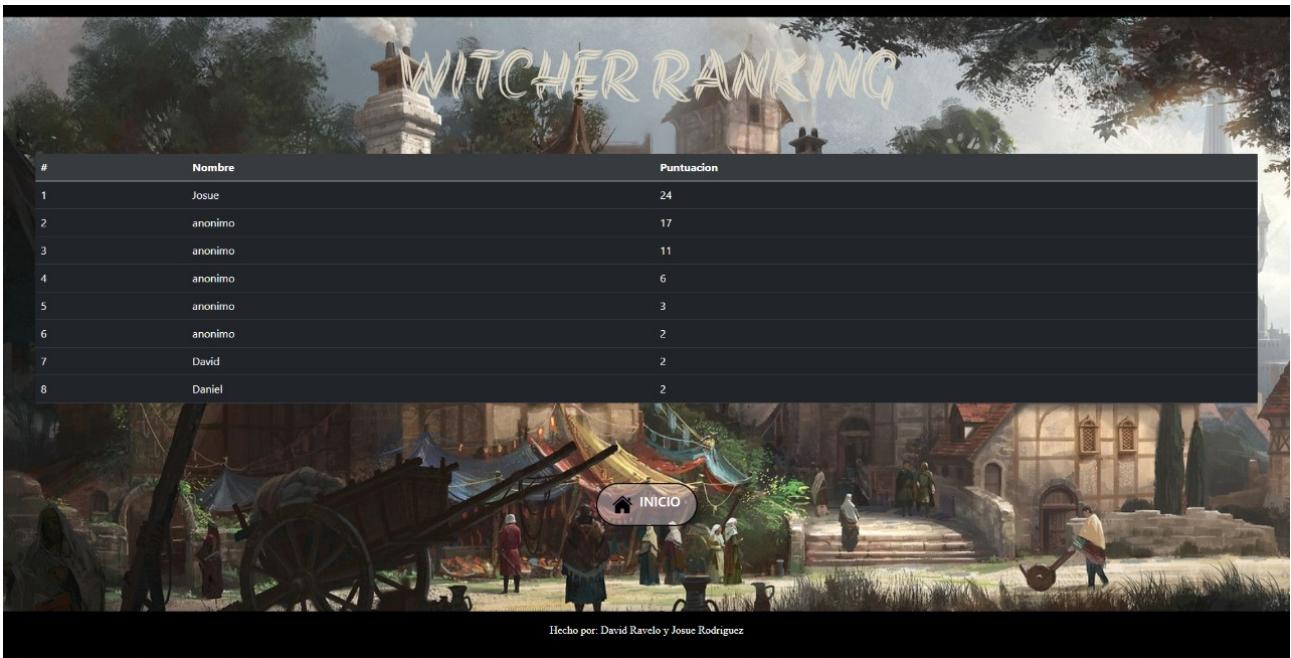


Imagen [5] Ranking.

6.2. Funcionamiento Técnico.

Para el funcionamiento del juego, se crean unos scripts, los cuales están modularizados, que cumplen un rol específico. Cada uno de ellos interactúa con los demás mediante relaciones de composición y asociación.

Scripts:

- *app.js*: El script de app se encarga de obtener el DOM del html donde se llama dicho script, lo cual permite generar el entorno para dibujar el tablero del juego que se obtiene del script *board.js*; Además de esto, allí es donde se genera el evento de escucha del teclado.
- *board.js*: Este script se encarga de renderizar todas las imágenes, textos y componentes del juego, dependiendo de la situación que se pueda presentar en el juego como: pausa, colisión y fin del juego.
- *Laser.js* y *Mago.js*: Estos scripts son los protagonistas del juego, de modo que en estos se tienen las instrucciones de dibujar cada uno de ellos, de ir actualizando su posición de acuerdo al desarrollo del juego y para el caso de *Mago.js*, se tiene también la instrucción o método de detectar colisión; En concreto, los siguientes métodos:
 - Métodos comunes:
 - *draw*: Este método se encarga de renderizar la imagen del juego en la posición que le corresponda, recibiendo como parámetro un *ctx* el cual es el contexto sobre el cual dibujar.

- update: La función del método update es de actualizar frame por frame la posición del laser, de modo que se perciba el movimiento y el desplazamiento de este mismo.
- *Laser.js*:
 - modifyVelocidad: Este método modifica la velocidad del laser, la cual irá aumentando respecto a la variable de complejidad de *complexity.js*.
 - reset: Mediante este método se restablecen los valores iniciales de la posición y la velocidad del laser.
- *Mago.js*:
 - colision: El método de colision tiene como finalidad detectar el enfrentamiento entre un laser y el mago mismo, de este modo se permite conocer si dicho proyectil hizo contacto con el personaje.
 - move: este método es el encargado del desplazamiento del mago, recibiendo como parámetro una orden que indica un movimiento hacia arriba o hacia abajo.
- *complexity.js*: Dicho script se encarga de controlar la complejidad del juego, la cual va a ir aumentando respecto al puntaje que se vaya adquiriendo. En él se encuentran las siguientes funciones:
 - *increaseVelocity*: Incrementa la velocidad de los láseres.
 - *increaseLasers*: Incrementa la cantidad de láseres en el juego.
 - *modifyComplex*: varía el tope de complejidad dependiendo del puntaje del juego.

Templates:

Los templates proveen las estructuras html sobre el cual se desarrolla el juego. Allí se crean cuatro templates en específico: *index.html*, *instrucciones.html*, *juego.html* y *ranking.html*; Cada una de estas, muestra una vista detallada y única para cada sección del juego, además de que cada template representa una página distinta.

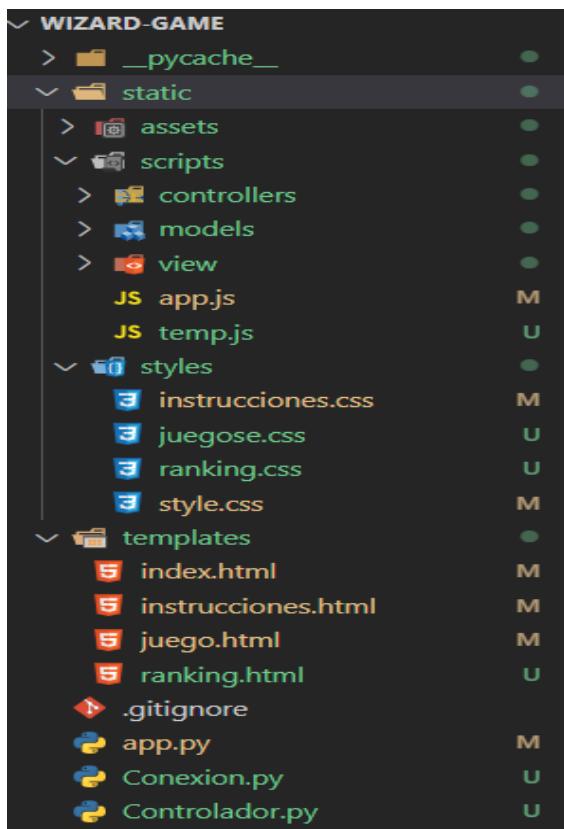
Styles:

Los estilos son el cuerpo de la página web mediante los cuales se da estética a las estructuras html. En este caso se crea un hoja de estilo en cascada para cada página o template siendo las siguientes: *instrucciones.css*, *juegose.css*, *ranking.css*, *style.css*.

MySQL + Python:

La base de datos es trabajada desde MySQL workbench 8.0 CE, la cual se conecta desde una base de datos por nombre puntuación desde conexión.py, el cual tiene una sola función llamada obtenerC(), la cual se conecta a la base de datos y devuelve esta conexión para poder hacer uso de sus tablas, la tabla record sera la única existente, que en este caso será usada para guardar puntajes y mostrar el record de cada uno de los jugadores ingresados.

6.2. Estructura de módulos y/o paquetes del proyecto:



Directarios:

- static: Directorio donde se guarda el contenido estático del juego.
- assets: Directorio que almacena las imágenes del juego.
- scripts: Directorio donde se encuentra la parte funcional del juego hecho con javascript.
- styles: Directorio que almacena el css del juego.
- templates: Directorio que guarda las estructuras html del juego.

Imagen [6] Paquetes.

6.3. Paradigmas de programación

6.3.1. Imperativo:

El paradigma imperativo se puede prever a la hora de declarar funciones y definir comportamientos específicos para realizar una tarea definida. En el proyecto se puede presenciar en las siguientes secciones de código:

- Al definir las consultas a la base de datos

```

Controlador.py ▶ s/s/app.js >
1 from pymysql import NULL, STRING
2 from pymysql.cursors import Cursor
3 from Conexion import obtenerC
4
5 def insertar_puntaje(nickname="anonimo",puntaje=0):
6     conexion = obtenerC()
7     if(len(nickname) == 0):
8         nickname = "anonimo"
9     with conexion.cursor() as cursor:
10        cursor.execute("INSERT INTO record(Nickname,Puntaje) VALUES (%s,%s)", (nickname,puntaje))
11        conexion.commit()
12        conexion.close()
13 def obtener_tabla():
14     conexion=obtenerC()
15     tabla=[()]
16     with conexion.cursor() as cursor:
17        cursor.execute("SELECT Nickname,Puntaje from record;")
18        tabla = cursor.fetchall()
19
20     conexion.commit()
21     conexion.close()
22 l=list(tabla)

```

Imagen [7] Controlador-imperativo.

- Al definir el script que controla el comportamiento de los componentes

```

Controlador.py ▶ s/s/app.js
>> 1 import Board from "../scripts/view/board.js";
2 █
3
4 var ctx = document.getElementById("myCanvas").getContext("2d");
5 var tecla;
6 var teclaPause = true;
7 var puntaje_contenedor = document.getElementById("puntaje");
8 var canvas = document.getElementById("canvas_container");
9 var puntaje = document.getElementById("punt");
10 document.addEventListener("keydown", handleKey);
11
12
13 function handleKey(e){
14     switch (e.keyCode){
15         case 87:
16             tecla = 'arriba';
17             break;
18         case 83:
19             tecla = 'abajo';
20             break;
21         case 80:
22             teclaPause = !teclaPause;
23             break;
24     }
25     e.preventDefault();
26 }
27
28 var colision = false;
29 var board = new Board(ctx);
30 const drawScenary = () =>{
31
NORMAL ▶ static/scripts/app.js                                     javascript

```

Imagen [8] App - imperativo.

6.3.2. Orientado a objetos:

El paradigma orientado a objetos se utilizó para definir los componentes del juego, creando clases y designando sus respectivos atributos y métodos, tal como se evidencia a continuación

```

s/s/m/Laser.js ➤
>> 1 export default function Laser(){
2     this.posX = 240;
3     this.posY = Math.floor(Math.random()*(145)); // (max - min) + min;
4     this.img = document.getElementById('laserGreen');
5     this.velocidad = 5;
6
7
8     this.draw = function(ctx){
9         ctx.drawImage(this.img,this.posX,this.posY,50,5);
10    }
11
12    this.modifyVelocidad = function(velocidad){
13        this.velocidad+= velocidad;
14    }
15
16    this.reset = function(){
17        this.velocidad = 5;
18        this.cambioVelocidad = 5;
19    }
20
21    this.update = function(ctx, colision){
22        this.draw(ctx);
23        if(this.posX < -50 || colision){
24            this.posY = Math.floor(Math.random()*(145)); // max 145
25            this.posX = 240 ;
26            if(colision)
27                this.reset();
28        }
29        else{
30            this.posX = this.posX-this.velocidad;
31        }
NORMAL ➤ static/scripts/models/Laser.js

```

Imagen [9] Laser - P.O.O.

6.3.3. Scripting:

El scripting se implementó con el uso de html, css y javascript, así como se ve en los siguientes fragmentos de código:

- En los distintos templates html.

```

s/s/m/Laser.js ▶ t/index.html
10     <link rel="preconnect" href="https://fonts.googleapis.com">
11     <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
12     <link href="https://fonts.googleapis.com/css2?family=Akronim&display=swap" rel="stylesheet">
13
14 </head>
15 <body>
16     <header>
17         <div class="container-header" >
18             <h1>WIZARD GAME<h1>
19         </div>
20
21     </header>
22
23     <div class="Botones">
24         <a href=game>
25
26             <button type ="button" class="Boton_1">
27                 
28                 <P class="p_boton">JUGAR</P>
29             </button><br>
30         </a>
31         <br>
32         <a href="instrucciones">
33             <button type="button" class="Boton_2">
34                 
35                 <P class="p_boton">INSTRUCCIONES</P>
36             </button>
37         </a>
38         <br>
39         <a href="ranking">
40             <button type="button" class="Boton_3">
NORMAL ➤ templates/index.html                                     html ◀ uti

```

Imagen [9] Index - scripting.

- En las diferentes hojas de estilo CSS

```

s/s/style.css ➤
1 body{
2     background-image: url('../assets/3629.jpg');
3     background-repeat: no-repeat;
4     background-size: cover;
5     min-height: 100vh;
6     font-family: 'Medula One', sans-serif;
7 }
8
9 .container-header{
10     margin-top: 50px;
11 }
12 .container-header h1{
13     color: #e0d9c6;
14     text-align: center;
15     font-family: 'Akronim', cursive;
16     font-size: 100px;
17 }
18 .Botones{
19     position: absolute;
20     top: 50%;
21     left: 50%;
22     transform: translate(-50%, -50%);
23 }
24
25
26
27 .Boton_1{
28     text-decoration: none;
29     padding: 10px;
30     font-weight: 600;
31     font-size: 20px;
NORMAL ➤ static/styles/style.css

```

6.3.4. Funcional - Declarativo:

Se empleó el paradigma funcional, especializado por la rama declarativa, para aplicar la complejidad sobre los láseres, y también para aumentar la cantidad de láseres en el mapa. Esto se puede evidenciar a continuación:

```
48     this.arrLaser.map((e)=>{ e.update( this.ctx, this.colision)});  
49 //complejidad del juego-----
```

Imagen [11] Map-laser-funcional.

```
6  
7     increaseVelocity(listLaser, timeScore){  
8         if(timeScore > this.cambio && this.cambio <= 30){  
9             listLaser.map((laser)=>{  
10                 laser.modifyVelocidad(2);  
11             });  
12         }  
13     }
```

Imagen [12] Map-complexity-funcional.

También se aplica el paradigma mencionado anteriormente, para obtener la lista de puntajes de la Base de Datos, la cual se ordenará mediante una función sorted que posee python, obteniendo una lista ordenada descendentemente.

```
13 def obtener_tabla():  
14     conexion=obtenerC()  
15     tabla=[()]  
16     with conexion.cursor() as cursor:  
17         cursor.execute("SELECT Nickname,Puntaje from record;")  
18         tabla = cursor.fetchall()  
19       
20     conexion.commit()  
21     conexion.close()  
22     l=list(tabla)  
23     return sorted(l,key=lambda puntaje:puntaje[1],reverse=True)
```

Imagen [13] sorted-conexion-funcional.

6.3.5. Back-end:

Del lado del backend se usa Flask, que permite generar rutas y establecer las conexiones con la base de datos, de manera que se puedan enviar y obtener datos de esta misma.

```

Controlador.py > s/s/app.js > s/s/c/complexity.js > s/s/style.css > s/s/v/board.js > app.py
1 from flask import Flask, render_template, redirect, abort, request
2 import Controlador
3
4 app = Flask(__name__)
5
6 @app.route("/")
7 def hello():
8     return render_template('index.html')
9
10 @app.route("/guardar_puntuacion", methods=['POST'])
11 def guardar_puntuacion():
12     nickname=request.form['Nickname']
13     puntaje=request.form['Puntuacion']
14     Controlador.insertar_puntaje(nickname,puntaje)
15     return redirect("/")
16
17 @app.route("/ranking")
18 def ranking():
19     tablas = Controlador.obtener_tabla()
20     return render_template('ranking.html', tablas=tablas)
21 @app.route("/instrucciones")
22 def instrucciones():
23     return render_template('instrucciones.html')
24
25 @app.route("/game")
26 def game():
27     return render_template('juego.html')
28 if __name__ == '__main__':
29     app.run(port=3000, debug= True)

```

NORMAL ➤ app.py

python ↶

Imagen [14] app-flask-backend.

7. Conclusiones

- Flask es una herramienta muy útil, al permitir el uso de bases de datos, el uso de python y de muchas otras herramientas para la creación de páginas web.
- El uso de python para el backend de una página web, facilita mucho la creación de cualquier funcionalidad, al ser python un lenguaje tan usado y con tantas herramientas y bibliotecas disponibles.