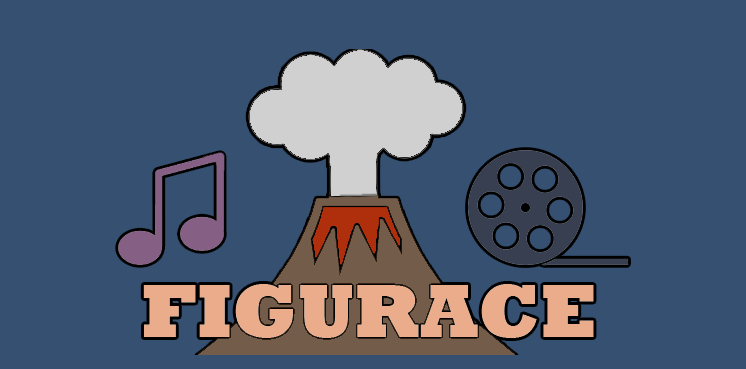
INFORME FINAL

*Documento explicativo sobre el desarrollo de la aplicación FiguRace*



**Antonio Glorioso Ceretti, Austin Myles, Fabrizio Torrico y Sebastian Ferro**

9 de agosto de 2022

Seminario de Lenguaje Python

## Índice

## [Introducción](#_wdtrsyo0w02h)

## [Temas Estudiados](#_s2xofiunjy5k)

1. [Problemas](#_103yzjhk1pjp)
   1. [Navegación](#qchk2ngxjmun)
   2. [Menú Principal](#hzazf1v3610k)
   3. [Perfil](#7daa5s6rlv5j)
   4. [Puntajes](#7vnoa6ra35s2)
   5. [Configuración](#4w7ss8f3urwv)
   6. [Juego](#eyxk7e9fiuem) (Incompleto)
   7. [UI](#j5w0ww2hye33)
   8. Pandas
   9. [Misceláneo](#86vqnbqi9him)
2. [Consideraciones éticas sobre el desarrollo](#_7kgyh7no8q0t)
3. [Conclusiones y trabajos futuros](#_chanyx93to50)
4. Anexo 1: Guía de usuario
   1. [Iniciar](#_ij5eysi7kimc)
   2. [Perfil](#_twcramljnwy8)
   3. [Puntaje](#_jzurspg42sda)
   4. [Configuración](#_d3u7an2wyk6j)
   5. [Jugar](#_5enepgh622az)
5. [Anexo 2: Guía de desarrollo](#_3hzcfm453r8y)
6. [Referencias](#_fvm8ob4weao7)

## Introducción

En este informe vamos a detallar la implementación, dificultades y razonamientos utilizados para construir nuestra aplicación: FiguRace.

Desde un principio tuvimos que plantearnos una forma organizada para trabajar en grupo mediante el uso de GitLab. Nos aprovechamos principalmente de su sistema de “issues” para separar los problemas en partes más pequeñas y manejables. Cuando alguno disponía de tiempo para programar simplemente seleccionaba un issue y comenzaba a intentar resolverlo.

También decidimos utilizar el modelo MVC (por sus siglas en inglés Model, View, Controller) ligeramente modificado para la organización del diseño de la aplicación, permitiendo una mayor legilibilidad del código y facilidad para extender el funcionamiento.

Toda la interfaz del proyecto está basada en la librería PySimpleGui, aunque conforme se fue complejizando el esquema tuvimos que añadir e implementar otras sobre la base (Pandas para la muestra de datos, PyGame para la implementación de música, etc.).

## Temas Estudiados

1. 1 **- PySimpleGUI**

Como bien lo expresa en su Readme *“PySimpleGUI es un paquete de Python que permite a programadores de Python de todos los niveles a crear GUIs”* (*Readme*, n.d.) [1](#ev61l4aj29ua) esta librería fue el esqueleto de la interfaz gráfica de nuestro juego y es la que utilizamos para poder desarrollar el diseño de la aplicación. Luego de trabajar con ella e investigar ciertas partes estructurales de las misma creemos que tenemos un conocimiento general sobre ella, permitiéndonos poder implementarla en futuros proyectos.

1. 2 **- Pandas**

Teniendo en cuenta la funcionalidad de esta librería *“pandas es un paquete de Python que provee estructuras de datos rápidas, flexibles y expresivas diseñadas para trabajar con datos “relacionados” o “etiquetados” de manera fácil e intuitiva”* (*Package Overview — Pandas 1.4.3 Documentation*, n.d.) [2](#xgzt1bfhpoe3) trabajamos con ella para el procesamiento de datos y análisis en profundidad de los mismos. Pudimos obtener ciertas características de las partidas jugadas y crear gráficos circulares mostrando gráficamente la información.

## Problemas

1. 1-**Navegación:** Debido a que PySimpleGUI no permite inyectar elementos mientras el juego se está corriendo, tuvimos que idear una forma sencilla para cambiar de ventanas. Considerando que nuestro proyecto era considerablemente pequeño y sencillo, decidimos utilizar un sistema de ocultamiento donde el programa guarda la referencia a un diseño de pantalla en un archivo de datos temporal y solo se mostrará la ventana necesaria. Cuando se cambia de escena, lo único que debemos hacer es actualizar la variable que guarda la ventana a mostrar y luego ocultar la ventana actual para que no se superpongan.
2. 2 - **Menú Principal:** El mayor problema con el cual nos encontramos fue el hecho de poder cambiar la información del jugador actual, dificultad actual y archivo con extensión csv elegido tal que dicha cambio persista durante toda la aplicación. Para resolverlo terminamos utilizando una función que guarda la información en un archivo temporal y luego nuestro archivo “model” (el cual controla la carga y distribución de los datos guardados) se actualiza en base al archivo temporal.
3. 3 - **Perfil:** Varias partes de la aplicación tendían a ser inestables ante la falta de al menos un usuario, por esto, decidimos forzar la pantalla de creación en el caso de que no existiera ninguno actualmente. La primera vez que se corre la aplicación, si esta no encuentra a ningún usuario guardado entonces abrirá la pantalla de creación de forma automática, ocultando el menú principal.
4. **4 -** **Puntajes:** Las dificultades surgidas en el diseño de la página de puntajes están relacionadas con el procesamiento de la información de puntos. Había que idear una forma de enlistar en orden y por dificultad todos los puntajes dentro del json, además de sacar el promedio total. Para ello programamos dos funciones (make\_score\_best(arg: scores), make\_score\_prom(arg: scores)) donde cada una recibe la listas y nos devuelve, en la primera, la lista ordenada, y en la segunda, el promedio de cada usuario.

*Make\_score\_prom* es la más compleja de las dos ya que necesita de una estructura auxiliar para ir sumando por cada dificultad y usuario la cantidad de puntos y partidas.

Posee un tiempo de ejecución de O() que no suele ser el más ideal, pero al tener una base de datos tan pequeña la carga de los mismos no presenta un problema.

1. **5 -** **Configuración:** La pantalla de configuración nos presentó dos grandes problemas: ¿Como y donde guardar la información? y ¿Cómo validar y actualizar la información guardada en tiempo real?.

La información decidimos guardarla en un archivo JSON para que persista para futuras sesiones. De esta forma, lo único que el programa debe hacer es leer los campos del archivo JSON al ejecutarse para tener todos los datos que el juego necesite.

La validación de los campos fue lograda mediante funciones para detectar el tipo y campo específicos donde surgió un error. Aquí nos aseguramos de no permitir ciertos nombres (como por ejemplo un nombre que solo sean espacios) y de avisar al usuario en caso de que los campos ingresados sean inválidos.

La actualización de la información guardada se logra mediante métodos de PySimpleGui.

1. **6 -** **Juego:** Obviamente, al ser uno de los aspectos principales de la aplicación, esto fue lo que más problemas nos ocasionó. Debido a que casi toda la información guardada por el programa es utilizada en esta pantalla, cuando comenzamos su construcción surgieron varias dificultades: había lectura incorrecta de datos, no se actualizaban los datos guardados, el temporizador no funcionaba o funcionaba de forma incorrecta, etc.

Además, se debían guardar datos para su estudio mediante el uso de pandas, lo que agregó otra capa más de complejidad al sistema y presentó nuevos problemas: Los datos no se guardaban, los datos guardados eran incorrectos, ciertos datos no se guardaban pero otros sí, ciertas situaciones rompían el almacenamiento de datos, etc.

1. **UI:** En un principio al no conocer tanto las herramientas de PySimpleGUI, la UI del juego se mantuvo algo básica, sin ningún icono o imagen en la pantalla de título. Una vez se descubrieron estas herramientas que dispone el contenedor, pudimos mejorar el aspecto general de la aplicación, perfeccionando las opciones del juego, añadiendo una imagen representativa para el título del juego y iconos para las opciones de “Salir del Juego”, “Menú de Inicio”, “Mutear Música” y “Desmutear Música”. Por lo demás la UI durante el gameplay se mantuvo simple para no confundir al jugador y que no tenga muchas cosas en pantalla durante su sesión de juego. Las opciones, la pantalla de Puntajes y de Perfiles al igual que la de Juego se mantuvo simple.

1. **Pandas:** Tuvimos algunos problemas de interpretación sobre qué información se debía guardar y procesar mediante Pandas. Pero una vez que nos pusimos de acuerdo sobre cómo solucionar las consignas las únicas dificultades fueron aprender a utilizar la librería.

1. **Misceláneo:** Aquí dejamos algunos otros problemas surgidos a lo largo del desarrollo del programa que no entran en ninguna categoría en particular:
   * **Música:** El sistema de música originalmente se encontraba en el mismo archivo que maneja la configuración del programa, pero finalmente decidimos dejarlo en su propio archivo a ser manejado y utilizar un sistema de clases para que sea más simple su actualización y manejo desde cualquier pantalla del programa.
   * **Iconos:** No fue un problema mayúsculo pero estaba presente el hecho de tener que transformar una imagen de formato .png a Base64 se hizo a momentos engorroso, ya que esto es necesario para que los iconos pueden mostrarse en pantalla y que PySimpleGUI crashee.
   * **Imagenes:** Aunque simple, el único inconveniente que hubo fue la configuración del path o ruta.
   * **Portabilidad:** Tuvimos que instalar Ubuntu en una máquina virtual para asegurar que el programa funcionase en Linux.

# Consideraciones éticas sobre el desarrollo

Nuestra aplicación es de licencia libre y código abierto, con todas sus dependencias cumpliendo esta característica. Puede ser usada por cualquiera, ya sea con motivos lúdicos o económicos. Motivamos la exploración del código fuente y la modificación del mismo para mejorar la experiencia del usuario o extender características del juego mismo.

Los sistemas operativos soportados y testeados por el software fueron Windows y Linux, no nos responsabilizamos por mal funcionamiento en cualquier otro.

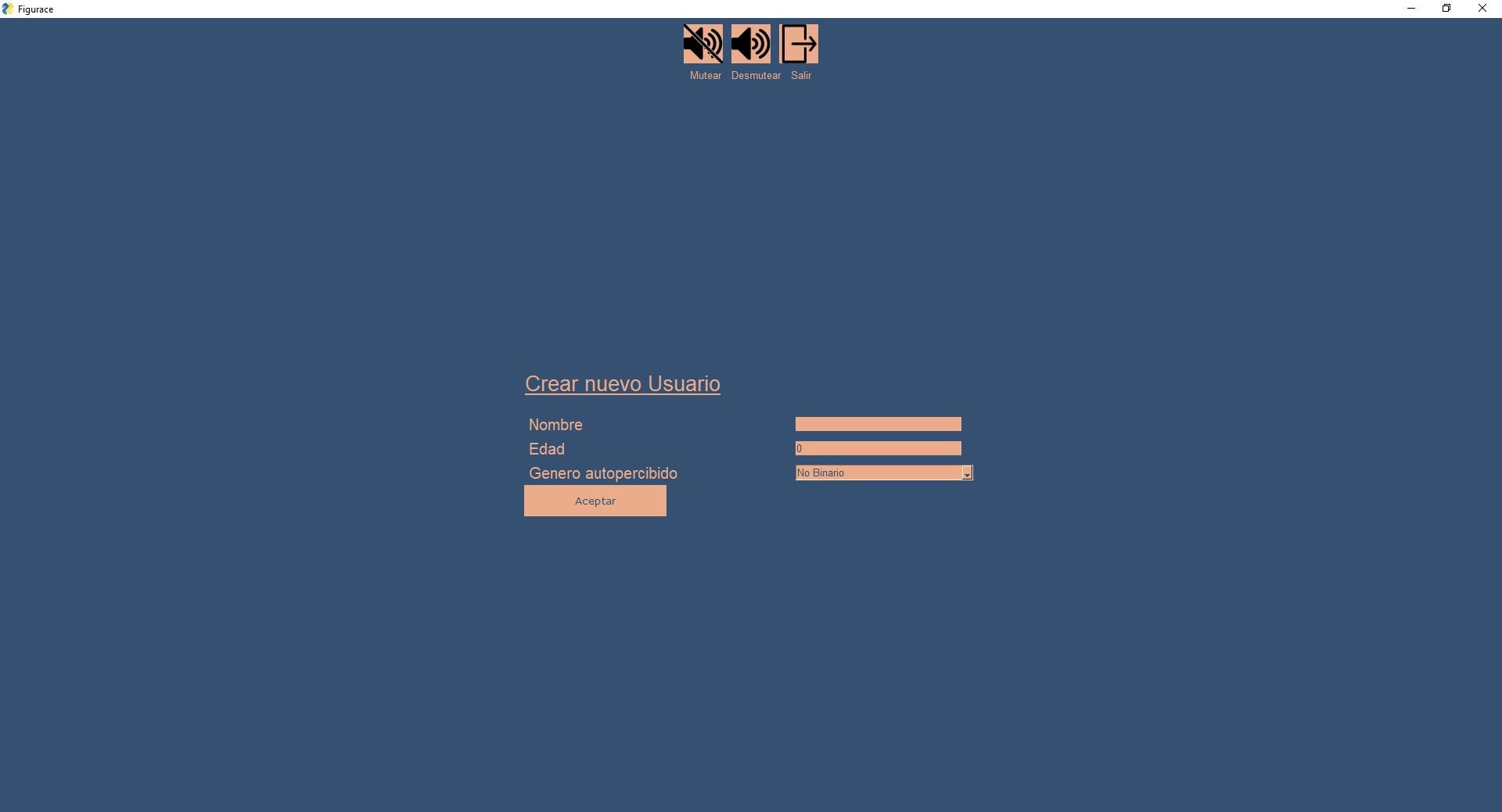
# Conclusiones y trabajos futuros

Realizar este trabajo nos permitió en primer lugar investigar e informarnos sobre herramientas integrales para la creación de software escrito en Python, ya sea en su diseño (con PySimpleGUI) o en el análisis de datos (con Pandas). Y además experimentar un ambiente de trabajo grupal donde debemos explayar nuestras ideas y compartirlas, escuchando así mismo las de los demás compañeros, preparándonos para desarrollos futuros.

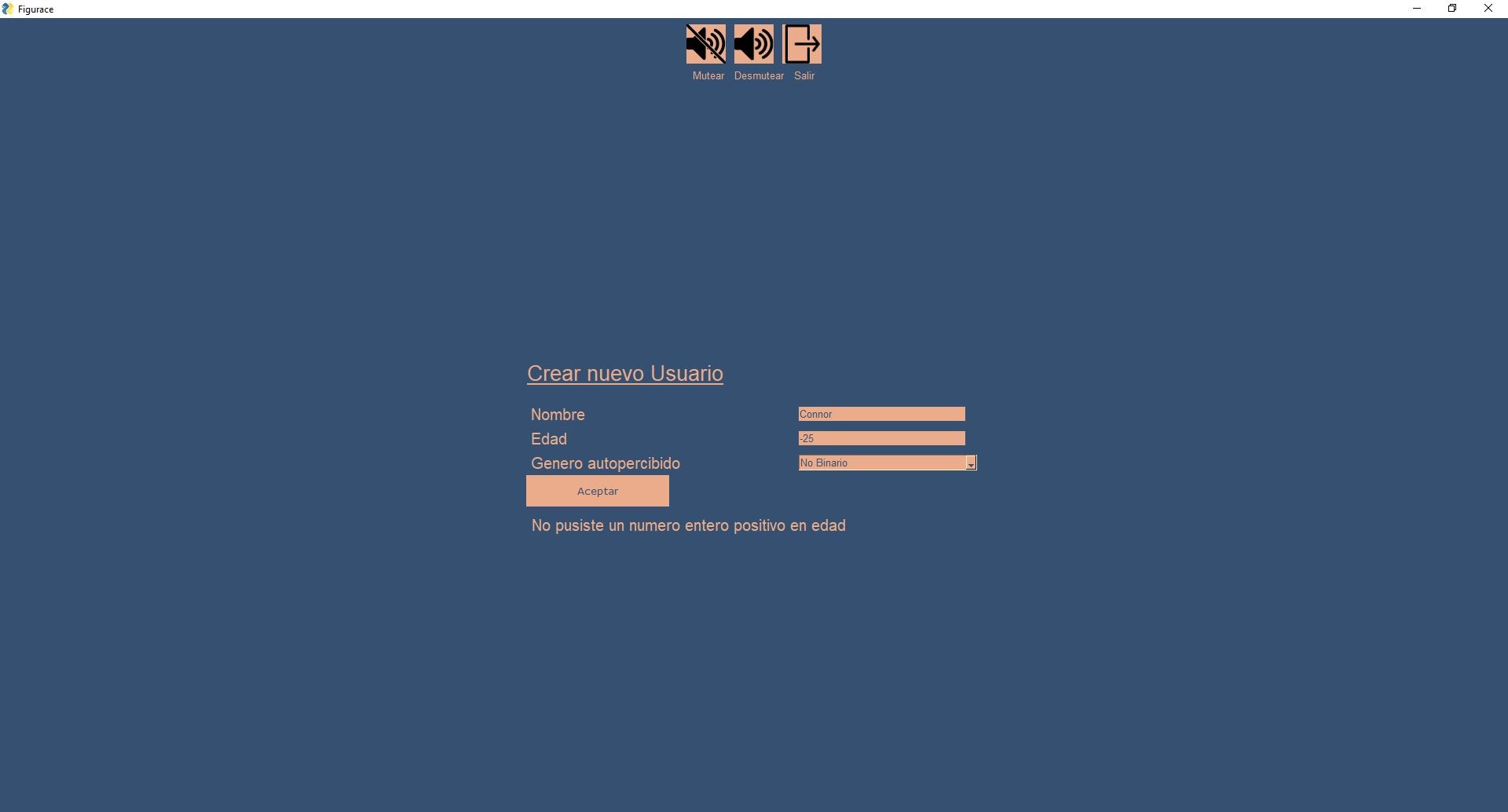
## Anexo 1: Guía de usuario

### Iniciar la aplicación:

Al iniciar la aplicación (y asumiendo que todavía no existe ningún usuario creado), el jugador se encontrará con la siguiente pantalla para crear un usuario nuevo:



El programa se asegura de que no pueda crearse un usuario con nombre vacío, pero decidimos permitir que el nombre sea no convencional (por ejemplo, un solo carácter o puros números). Además, la edad solo permitirá guardar números positivos, en caso de que se introduzcan números negativos o caracteres, al intentar guardar el usuario el mismo programa avisará del error, como es el caso de la siguiente figura:

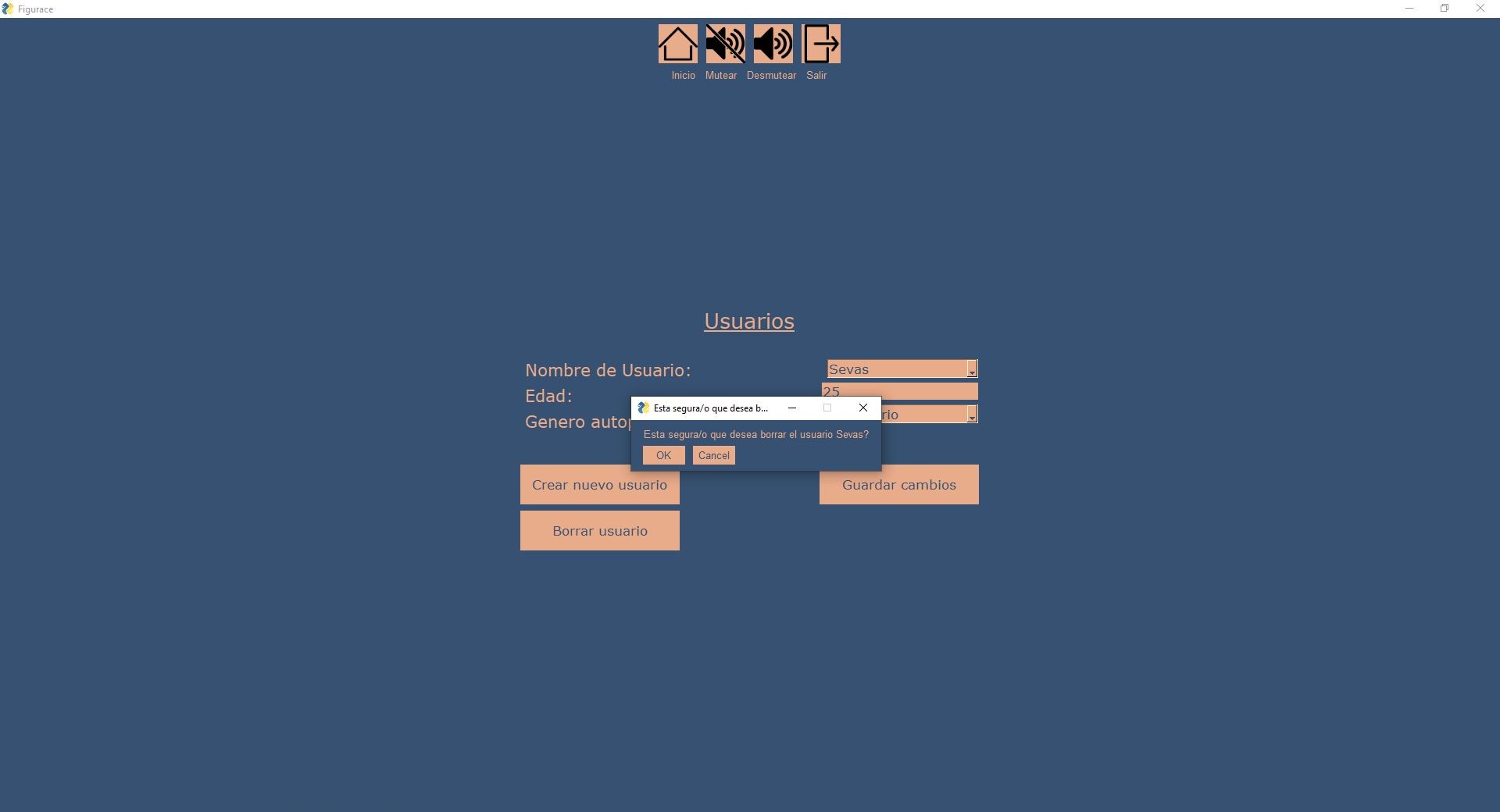


El menú principal cuenta con tres “componentes”. Un grupo de cuatro botones en el borde superior de la aplicación (los cuales son accesibles desde cualquier ventana del juego y brindan la capacidad de mutear o desmutear la música, volver al menú principal y salir de la aplicación).   
La barra de navegación en el centro, con cuatro botones para navegar entre las distintas ventanas del juego (Jugar para comenzar con el juego, Configuración para cambiar las opciones del mismo, Puntaje para ver los mejores puntajes de cada dificultad y csv y finalmente Perfil para las opciones del perfil).   
El último componente son los tres menús desplegables los cuales permiten seleccionar la dificultad, el usuario y el dataset con los cuales se va a jugar.

### Perfil:

Comenzando con las opciones de abajo hacia arriba, si seleccionamos el botón de “Perfil” tendremos acceso a la pantalla de perfiles, donde podemos cambiar los datos de nuestro perfil, borrarlo y también crear nuevos perfiles.

**Borrar un Usuario:**

****

Como podemos ver, el juego se asegura de preguntarnos una vez más si estamos realmente seguros de que queremos borrar un usuario, para que esto no suceda por accidente.

La pantalla de creación de perfil ya fue vista en la primera figura, la única diferencia es que si existe al menos un usuario cuando se quiere crear otro, entonces en los botones de navegación en la parte superior de la aplicación mostrará la opción de volver al menú principal.

**Sin Usuarios**:

****

**Con Usuarios**:

****

Esto es porque el juego jamás debería ejecutar el menú principal sin al menos un usuario, por esto, si no hay usuarios disponibles no se podrá navegar al menú principal hasta haber creado al menos un usuario.

### Puntaje:

La pantalla de puntajes es simple: Se muestran tres opciones de dificultad (Fácil, Medio y Difícil) y para cada opción se actualiza la lista con los 20 mejores puntajes y los 20 usuarios con el mejor promedio.

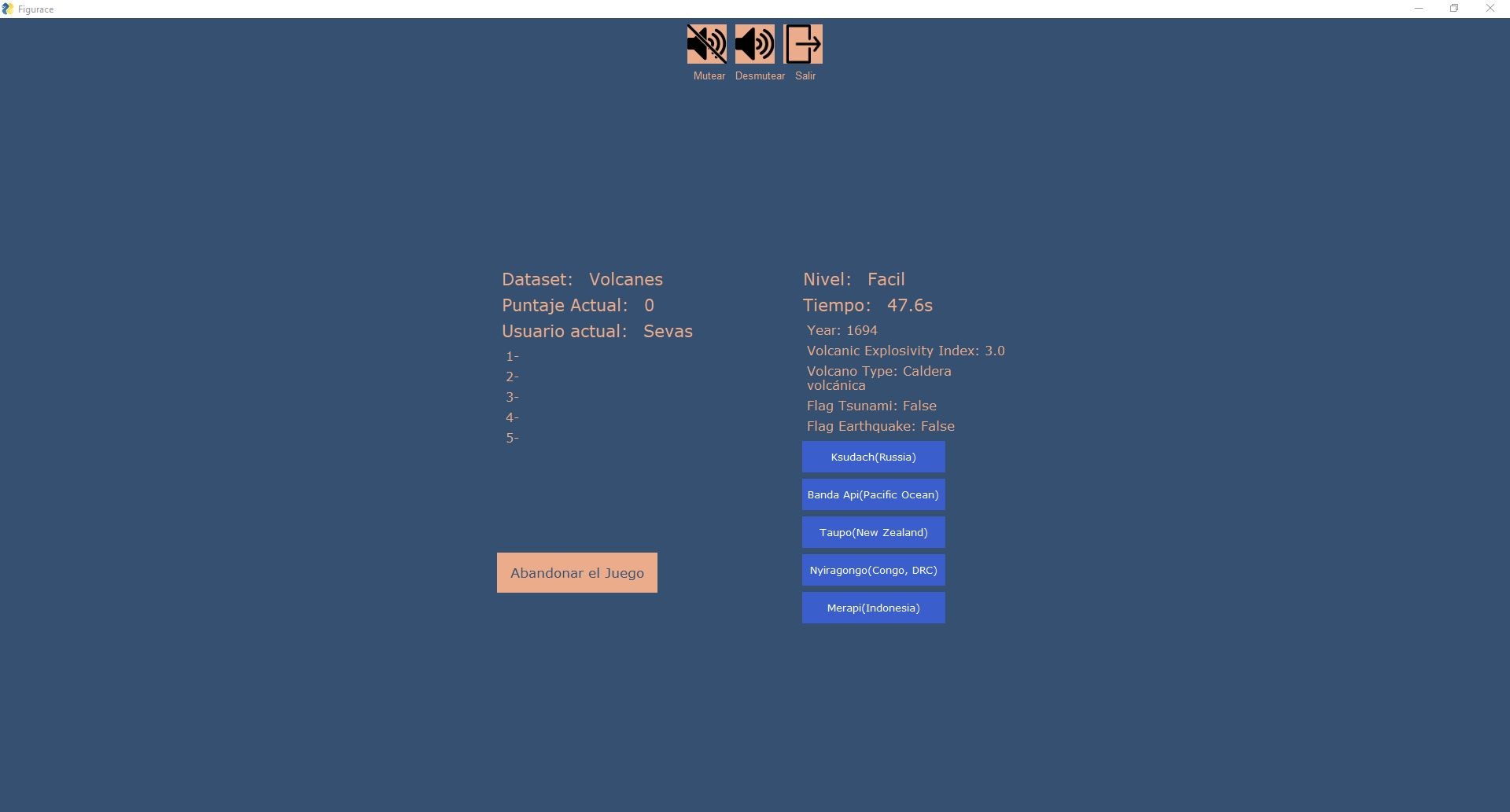
### Configuración:

Esta pantalla se encarga de establecer las varias opciones del juego (cantidad de rondas, tiempo por ronda, etc.). También controla el volumen de la aplicación mediante un slider.

Notar también cómo los cambios se efectúan para TODOS los niveles de dificultad, con la excepción de la cantidad de características por nivel, las cuales son dependientes del nivel de dificultad (en este caso Fácil). Esto se debe a que no tendría sentido poder configurar los niveles de dificultad de forma individual, ya que uno podría darse 200 segundos para responder las preguntas en el nivel de dificultad más alto, volviéndo el nivel de juego más fácil.

### Jugar:

Finalmente, tenemos la pantalla de juego. Aquí se nos presenta una pantalla con información variada sobre el dataset elegido (incluyendo las pistas para seleccionar una opción correcta), el tiempo restante para seleccionar una opción y cinco opciones posibles. El juego consiste en adivinar qué opción es la correcta en base a la información dada. En el ejemplo a continuación, podemos ver el año en el que fue descubierto el volcán, su índice de explosividad, su tipo y si generó Tsunamis o terremotos:



Cabe destacar que se muestran cinco “pistas” o “características” del volcán debido a que la dificultad es Fácil, en Medio y Difícil, la cantidad de características mostradas será menor.

Una vez seleccionada una opción, la pantalla mostrará la opción en verde (en caso de que sea correcta) o roja (en caso contrario). Además, en la lista a la izquierda del programa se mostrará la cantidad de respuestas correctas e incorrectas logradas hasta el momento. Un último detalle a tener en cuenta es que el temporizador cuenta el tiempo tardado para cada ronda, no para cada sesión de juego, por lo que al seleccionar una opción (sea correcta o incorrecta) el temporizador volverá a su estado original. Finalmente, al finalizar la sesión de juego, el programa redirigirá al usuario a la pantalla de Puntajes automáticamente.

## Anexo 2: Guía de desarrollo

### Patrón MVC: El patrón utilizado en el flujo de datos es el siguiente: Cada pantalla exporta 2 funciones principales: El controlador y la vista. Vista: Contiene el diseño y eventos enlazados a cada componente(botones, inputs, etc). Controlador: Contiene la lógica a realizar en cada evento, principalmente se encarga de actualizar los datos en el modelo y en el layout (Vista) Modelo: Contiene la información relevante que se puede actualizar en algún momento de la aplicación, también tiene métodos para manejar la información EJEMPLO: Un usuario puede levantar un evento mediante una acción (por ejemplo tocar el botón de cambiar usuario), este evento se toma en el controlador específico, encargado de actualizar en el modelo el usuario actual y en la ui los nuevos correspondientes valores

### Estructura de Archivos:

#### **Carpeta Principal:** La carpeta del juego contiene tres archivos de texto (el Readme, License y Requirements necesarios para que el programa funcione correctamente), la carpeta **src** que contiene todos los archivos pertinentes al funcionamiento del programa y finalmente el archivo **Figurace.py** (cuya unica funcionalidad es cargar el archivo **controller.py** desde la carpeta **src** para que comience a ejecutarse el programa).

#### **src:** La carpeta **src** (o “source”) contiene tres carpetas llamadas **common**, **data** y **windows**, donde cada una contiene diferentes componentes vitales para el funcionamiento del programa. Además, aquí se encuentra el archivo **controller.py** que se encarga de crear y mostrar el menú principal del juego. Este archivo también carga el resto de las ventanas del juego (play\_page, config\_controller, scores\_controller, profile\_controller y new\_profile\_controller) para su futuro uso. Finalmente, el **controller** se encarga de cargar la clase **M\_Player** (o Music Player) para poder escuchar música a lo largo de la ejecución del programa.

#### **common:** La carpeta **common** contiene solo dos archivos .py: **components** y **helpers**. El archivo **components** engloba funciones utilizadas para el desarrollo de la UI del programa utilizando PySimpleGUI. De esta forma nos aseguramos que el estilo de la UI sea idéntico en todos los archivos del programa. El archivo **helpers** engloba funciones generales que pueden ser usadas desde cualquier punto del programa para resolver problemas generales. Por ejemplo, la función “check\_integer” nos sirve cuando queremos saber si un valor dado es un entero o no.

#### **data:** La carpeta **data** contiene seis carpetas: **files\_csv**, **files\_csv\_to\_load**, **files\_json**, **img**, **jupyter\_notebooks** y **music**. Y tres archivos: **icons.py**, **model.py** y **music\_player.py**

Las carpetas **files\_csv** y **files\_csv\_to\_load** contienen los archivos .csv que serán utilizados por el programa para su jugabilidad. Los archivos originales se encuentran en la carpeta **files\_csv\_to\_load**, mientras que **files\_csv** contiene los archivos .csv actualizados para ser utilizados por el juego y el archivo **round\_analisis.csv** el cual guarda la información sobre las partidas jugadas a ser utilizada para el procesamiento de datos mediante pandas.

La carpeta **files\_json** contiene tres archivos .json llamados: **config**, **scores** y **users**, los cuales guardan la información sobre la configuración del programa, los puntajes de los usuarios y los datos de cada usuario respectivamente.

La carpeta **img** guarda las imágenes utilizadas por la UI del juego.

La carpeta **jupyter\_notebooks** contiene tres archivos .ipynb llamados: **movies**, **spotify** y **volcanoes** pedidos `por la cátedra donde cada uno procesa un archivo .csv (dentro de la carpeta **files\_csv\_to\_load**) y explica cómo lo “limpia” para crear un nuevo archivo .csv (dentro de la carpeta **files\_csv**) a ser utilizado por el programa en su jugabilidad. Finalmente, en esta carpeta se encuentra el archivo **data\_analisis.ipynb**, el cual se encarga de procesar los datos guardados dentro del archivo **round\_analisis.csv** mediante el uso de Pandas para resolver las consignas planteadas por la cátedra.

La carpeta **music** guarda los archivos de música .mp3 utilizados por el programa. Toda la música utilizada fue adquirida de la pagina “<https://www.fesliyanstudios.com/>” y es royalty free (de uso libre).

El archivo **icons.py** solo contiene variables que guardan el código de los iconos utilizados en la UI ya que estos eran demasiado largos y entorpecían la lectura del código donde eran utilizados.

El archivo **model.py** contiene casi todo el código necesario para el almacenamiento y lectura de información del programa. Los archivos .json y .csv por ejemplo son procesados mediante este archivo.

El archivo **music\_player.py** contiene la clase **M\_Player** que se encarga de administrar el sonido y la música utilizadas por el programa.

#### **windows:** La carpeta **windows** contiene seis archivos terminados en “**\_window.py**” los cuales se encargan de crear la UI necesaria para una página en particular (menú principal, puntajes, configuración, etc.) y un archivo llamado **views.py** el cual es utilizado como “esqueleto” para crear todo el resto de las páginas del programa.

# Referencias

1. *Readme*. (n.d.). PySimpleGUI. <https://www.pysimplegui.org/en/latest/readme/#what-is-pysimplegui>

1. *Package overview — pandas 1.4.3 documentation*. (n.d.). Pandas. https://pandas.pydata.org/docs/getting\_started/overview.html