Capítulo 1

Sistema propuesto

1.1. Setup a utilizar

Debido a que este trabajo de memoria corresponde fundamentalmente a una prueba de concepto se decidió junto al profesor guía evitar en lo posible la compra de nuevo equipamiento y por tanto no se entrega particular importancia a cumplir de forma estricta con las características de hardware utilizado en instancias formales de investigación. Con esto en consideración se presenta a continuación una descripción del entorno de hardware y software utilizado:

Hardware	PC	Procesador:	Intel Core i7-3770
		Memoria RAM:	16.0GB, DDR3, 650MHz
		Disco duro:	Samsung SSD, 840 Series, 120GB
		Tarjeta gráfica:	NVIDIA GeForce GTX 660 Ti, 2GB
	Monitores	Principal:	Samsung SyncMaster, 1680x1050px, 60Hz
		Secundario:	Samsung SyncMaster, 1680x1050px, 60Hz
	Adquisición	Eyetracker:	EyeTribe
Software	Entorno	SO:	Windows 10 Pro x64
		Base:	Python 2.7 x86, Anaconda
	Módulos	Principal:	PsychoPy 1.84.2
		Requeridos:	Ver anexo **??**

Cuadro 1.1: Hardware y software utilizado en el desarrollo.

La elección de software se encuentra infuenciada fuertemente por los siguientes puntoss:

1. El eyetracker a utilizar solo tiene disponible sus drivers para Mac y Windows. Debido a la facilidad de encontrar computadoras con

2. Los desafíos del sistema a implementar requieren del uso de un lenguaje orientado a objetos. A pesar de las bondades de Octave o Matlab en procesamiento y acceso a módulos complementarios estos entornos carecen de buen soporte para este tipo de programación, por este motivo se utiliza Python como lenguaje de desarrollo.

3. Dentro de las opciones disponibles PsychoPy resulta sumamente interesante. Cumple con ser open-source, es ampliamente utilizado por investigadores, presenta documentación detallada y clara para todas sus funciones y los foros se encuentran activos. Otro punto interesante es que entrega soporte para varias marcas de eyetracker, entre los cuales se encentra el seleccionados.

1.2. Diseño del sistema

El sistema a ser desarrollado en ente trabajo de título tiene como función principal facilitar el proceso de configuración y puesta en marcha de experimentos asociados a movimiento ocular. Para esto, se considera oportuno subdividir el desarrollo en tres partes: La primera consiste en determinar la estructura de datos requerida para almacenar tanto las configuraciones del sistema como de las tareas a implementar. La segunda implica la implementación de los métodos de configuración y ejecución del experimento, asegurando su correcto funcionamiento. La tercera etapa corresponde a montar estas funciones en una GUI para facilitar el proceso a personas que no tengan conocimiento del lenguaje. Así, se presenta a continuación el trabajo realizado.

1.2.1. Primera parte: Estructura de datos

Al diseñar la estructura de datos se debió considerar el problema desde tres aristas. La primera correspondía a la necesidad de almacenar la información del conjunto compuesto por experimento-tareas-cuadros-componentes donde, complementando lo expuesto en el apartado **??**, las tareas corresponden a las actividades específicas que el sujeto de prueba debe llevar a cabo. Los cuadros corresponden a los elementos que conforman dichas tareas y los componentes serán considerados como las figuras e imágenes contenidas en cada cuadro. La segunda dice relación con la información complementaria que se desea incluir para posterior análisis, tal como comentarios del investigador encargado del experimento para una sesión específica o algunas características de quien realizará el experimento, tales como su edad, sexo, color de ojos o si se utilizan lentes durante la ejecución. Finalmente, la tercera arista implica el conservar las configuraciones del equipo y hardware a utilizar para montar el servicio de ejecución asociado al módulo ioHub, que es el componente de PsychoPy que permite sincronizar todos los dispositivos asociados a la ejecución del experimento: teclado, pantalla, eyetracker, etc. y almacenar los datos en un archivo.

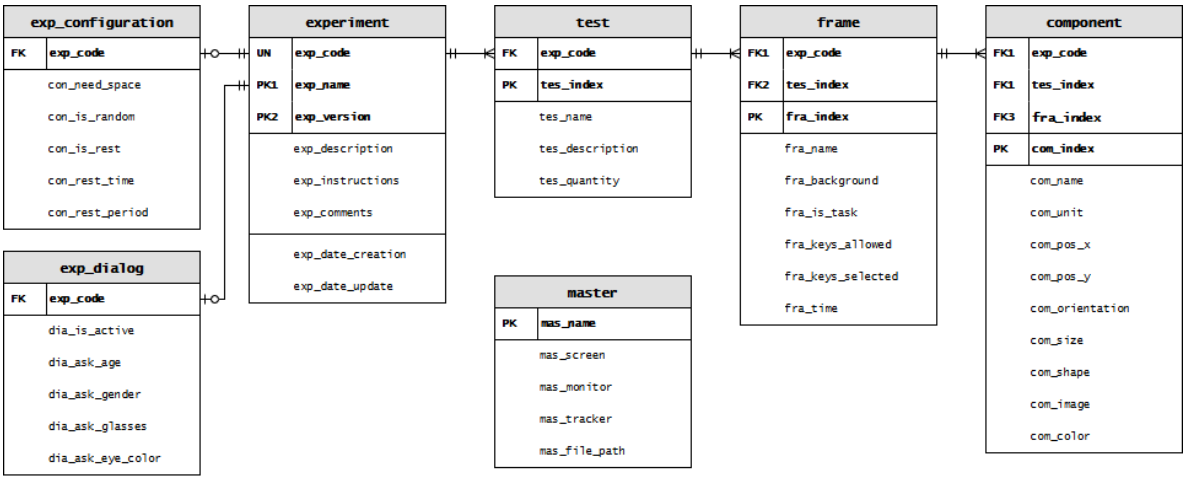

Figura 1.1: Diagrama de base de datos a implementar.

En base a lo anterior, se propone como solución el uso de una base de datos local de tipo SQLite con la estructura presentada en la figura 1.1. A continuación se comenta el contenido de cada tabla y su función:

1. **master:** Almacena la información asociada a un perfil de configuración del hardware a utilizar. Cada perfil es identificado con un nombre e indica que pantalla será utilizada (screen), cual es la configuración de dicha pantalla (monitor, se define en un aplicativo de PsychoPy), que eyetracker se utilizará (tracker) y en que directorio se almacenará el archivo resultante.

2. **experiment:** Almacena la información de identificación de un experimento. Cada experimento se identifica por un código único (code) y un par nombre/versión (name/-version), que tiene como objetivo identificar a una aplicación específica de una familia de el mismo tipo, por ejemplo: ”antisaccade”/”v1.0_gap”. Además, se incluye un campo para la descripción, instrucciones para el usuario (instructions) y comentarios para quien tome el experimento (comments). Se incluyen la fecha de creación y la última modificación para dejar de forma explícita una señal de cuidado: Las condiciones del experimento pueden no ser las mismas de la última ejecución.

3. **exp_configuration:** Esta tabla es complementaria al experimento y almacena la configuración general de ejecución. £Es necesario que el paciente presione la tecla espacio antes de cada tarea? (need_space), £el orden de las tareas es aleatorio o secuencial?

(is_random), £Se incluyen en el experimento tiempos de descanso? (is_rest), £Cada cuántas tareas? (rest_period), £Con qué duración? (rest_time).

4. **exp_dialog:** Esta tabla es complementaria al experimento y almacena la configuración del diálogo inicial. £Se incluirá información complementaria? (is_active), £Qué edad tiene el paciente? (ask_age), £Cuál es su sexo? (ask_gender), £Utiliza gafas o lentes de contacto? (ask_glasses), £Cuál es su color de ojos? (ask_eye_color).

5. **test:** Esta tabla contiene, para cada experimento, la identificación de las tareas a utilizar. Cada tarea se identifica por su nombre (name) y debe especificar el número de repeticiones a ser ejecutadas (quantity). Incluye un campo para añadir una descripción de la tarea (description).

6. **frame:** Esta tabla contiene, para cada tarea, el conjunto de cuadros que la conforman. Cada cuadro se identifica por un nombre (name) y permite la configuración de su comportamiento y características de forma general: color de fondo (color), si el cuadro corresponde a una tarea que requiere respuesta de teclado o es temporizada (is_task), las teclas habilitadas (keys_allowed), las teclas que se espera sean presionadas (keys_selected) y el tiempo por el cual debe ser presentada (time).

7. **component:** Esta tabla contiene, para cada frame, el conjunto de componentes o elementos que la conforman. Cada componente se identifica por un nombre (name) y las configuraciones asociadas a las unidades de medida consideradas (units), su posición en la pantalla (pos_x, pos_y), su tamaño (size), si se encuentra rotada o no (orientation), si es una figura, su forma (shape) y color (color) o si es una imagen el binario correspondiente (image).

Este modelo se considera apropiado por los siguientes motivos:

1. Aísla un experimento de otro al tener tareas y configuraciones completamente independientes. Esta configuración permite evitar que cambios en alguna tarea para ajustarse a necesidades específicas de un experimento afecte el funcionamiento del resto.

2. No puede existir una tarea que no se encuentre asociada a algún experimento, esto aplica también a cuadros y componentes.

3. Tener toda la información y configuraciones contenidas en un archivo facilita la portabilidad y migración desde un equipo a otro. Además, dado que es tratable como una base de datos convencional facilita el proceso de revisión de los datos sin necesidad del programa principal.

Cabe destacar que, para asegurar la limpieza de la base de datos se tomo la decisión de generar tanto updates como deletes en cascada a todas las tablas asociadas a un experimento, de esta forma, al eliminar un experimento particular se eliminan también todas sus tareas y configuraciones, evitando data residual. Esto se repite en niveles más bajos también: eliminar una tarea elimina todos sus cuadros, eliminar un cuadro elimina también todos sus componentes.

1.2.2. Segunda parte: Implementación de las funciones principales

Para lograr sincronizar la ejecución del experimento con el uso de dispositivos tales como el monitor, el teclado y el eyetracker se hará uso de ioHub. IoHub es un módulo que forma actualmente parte de PsychoPy y que fue desarrollado originalmente por Sol Simpson. Su funcionalidad principal consiste en montar un servicio por el cual se hace una revisión constante de los dispositivos de entrada/salida seleccionados, almacenando la información recopilada de forma ordenada en un archivo con formato hdf5, que corresponde a una tipo de diccionario donde se ordenan los datos. Para construir la aplicación en base a esta utilidad se debe generar un conjunto de métodos que permita lograr las funcionalidades presentadas en la figura 1.2.

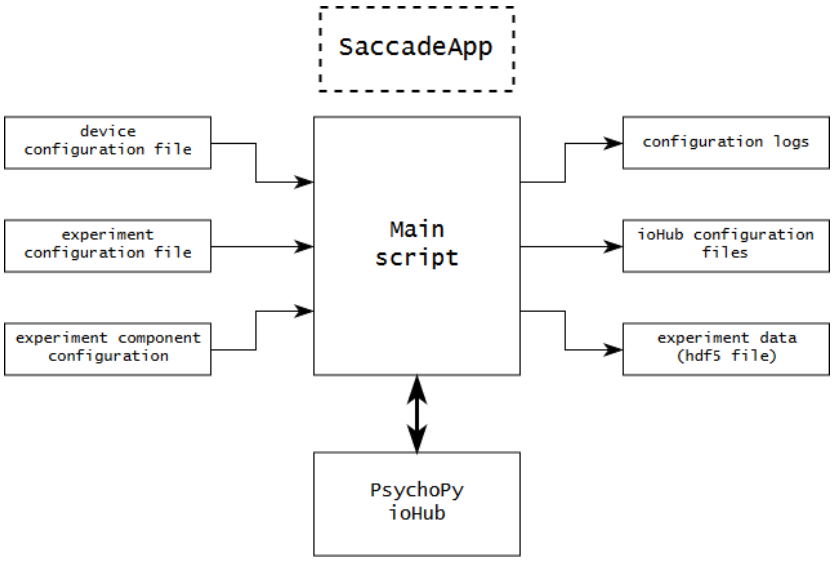

Figura 1.2: Diagrama de funcionamiento general.

Para esto, es necesario entregar tres insumos: la configuración de los dispositivos que serán monitorizados por ioHub, la configuración general del experimento (identificadores, descripción y los elementos que conforman las ventanas de diálogo) y la configuración de los elementos asociados a la presentación de estímulo (que se conforman por otras rutinas de PsychoPy).

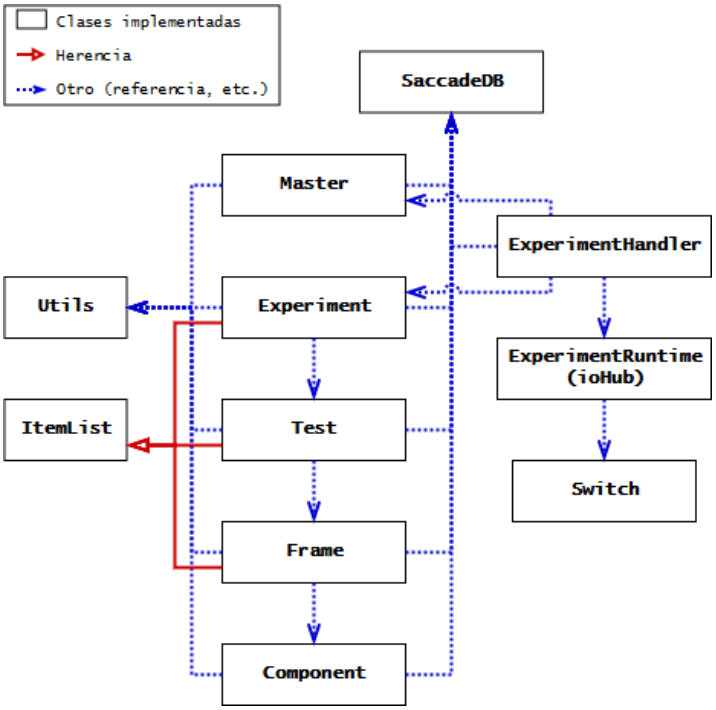

Figura 1.3: Diagrama de clases del sistema (no considera GUI).

En la figura 1.3 se presenta la estructura de clases implementada en este trabajo de título. Las clases Master, Experiment, Test, Frame y Component representan el conjunto de funciones que permiten manejar la configuración almacenada en las respectivas tablas de la base de datos además de otorgar otras funcionalidades que se enfocan en la ejecución del experimento propiamente tal. Debido a que Experiment, Test y Frame implican el manejo de una lista de componentes, estas son creadas como clases que heredan las funcionalidades básicas para el manejo de listas como el agregar, copiar o eliminar elementos, cambiar suu posición en la lista, etc. La clase utils implementa métodos de formateo de datos, tales como la conversión de strings a formato unicode o el formateo de fechas. SaccadeDB otorga funcionalidades que permiten comunicarse con el archivo de base de datos. Finalmente, la clase ExperimentHandler es la encargada de generar crear las carpetas de almacenamiento, los archivos de respaldo, los logs de configuración y la inicialización de la ejecución del experimento, que es implementado en la clase ExperimentRuntime (que hereda de ioHub e inicia el servicio asociado). La clase Switch implementa una estructura similar a un switch-case con el fin de utilizar una máquina de estados que regule la carga de frames durante la ejecución.

A continuación se describe con un mayor grado de detalle cada clase. Por simplicidad

se obvian las funciones asociadas al get-set de los atributos (que corresponden a los datos almacenados en la base de datos):

1. **Utils:** Clase utilizada para implementar funcionalidades de formateo de datos.

format_text	
Descripción	Método estático. Convierte un string o similar a formato unicode. Si el dato ingresado no cumple con las condiciones de formato se devuelve un elemento vacío.
Inputs	word: String o dato a ser formateado. lmin: Largo mínimo de la palabra. lmax: Largo máximo de la palabra.
Return	unicode.
format_int	
Descripción	Método estático. Convierte un valor en entero. Si el valor no cumple con las condiciones dadas se devuelve un valor por defecto, especificado por el usuario.
Inputs	value: Valor a ser formateado. vmin: Valor mínimo permitido. default: Valor por defecto.
Return	int.
format_float	
Descripción	Método estático. Convierte un valor en flotante. Si el valor no cumple con las condiciones dadas se devuelve un valor por defecto, especificado por el usuario.
Inputs	value: Valor a ser formateado. vmin: Valor mínimo permitido. default: Valor por defecto.
Return	float.
format_bool	
Descripción	Método estático. Convierte el valor entregado en un booleano. Si el valor no puede ser convertido se devuelve algún valor por defecto especificado por el usuario.
Inputs	state: Estado a ser formateado. default: Valor por defecto.
Return	bool.

format_path	
Descripción	Método estático. Formatea la dirección entregada dependiendo del sistema operativo.
Inputs	path: Dirección a ser formateada.
Return	unicode.
get_time	
Descripción	Método estático. Permite convertir un string asociado a una fecha desde el formato de tiempo GTM a el horario de Chile continental. Esta función se agrega debido a la configuración horaria de la base de datos.
Inputs	date: String que representa una fecha en formato Y-m-d H:M:S.
Return	unicode.

Cuadro 1.2: Métodos implementados en la clase Utils.

2. **SaccadeDB:** Clase utilizada para implementar la conexión con la base de datos SQLite.

SaccadeDB		
Descripción	Constructor. Inicializa los atributos del objeto.	
Inputs	filepath:	Ubicación (string o unicode) del archivo de base de datos.
Return	void.	
connect		
Descripción	Método que inicia una conexión con el archivo de base de datos. Si el archivo de base de datos epecificado en el constructor no existe, crea uno nuevo utilizando las configuraciones especificadas en **??**.	
Inputs	void.	
Return	void.	
close		
Descripción	Método que finaliza, de ser posible, la conexión con la base de datos.	
Inputs	void.	
Return	bool.	Verdadero si la conexión se cierra apropiadamente, falso en caso contrario.
push_query		
Descripción	Método que permite realizar querys que involucran modificar los contenidos de la base de datos (insert, update, delete, etc.).	
Inputs	query.	Instrucción SQL (string o unicode).
Return	bool.	Verdadero si la query se ejecuta correctamente, falso en caso contrario.

pull_query	
Descripción	Método que permite realizar querys que involucran obtener datos de la base de datos (select).
Inputs	query. Instrucción SQL (string o unicode).
Return	objeto. numpy_array en caso de existir datos, None en caso contrario.

Cuadro 1.3: Métodos implementados en la clase SaccadeDB.

3. **Master:** Clase que permite manipular las configuraciones de hardware asociado a los experimentos y la ruta de los archivos de salida de los mismos. Para utilizar las funciones de carga/guardado es necesario configurar el objeto de base de datos mediante la función set_database.

Master	
Descripción	Constructor. Inicializa los atributos del objeto.
Inputs	void.
Return	void.
get_list	
Descripción	Método estático. Entrega una lista con los nombres de los perfiles de configuración que se encuentran almacenados en la base de datos.
Inputs	db: Objeto de base de datos saccadeDB.
Return	objeto. list en caso de existir registros, None en caso contrario.
get_available_trackers	
Descripción	Método estático. Entrega una lista de las configuraciones disponibles para los eyetrackers. Dichas configuraciones pueden ser encontradas en la carpeta 'saccadeApp/resources/eyetrackers' del proyecto.
Inputs	void.
Return	list.
get_available_screens	
Descripción	Método estático. Entrega una lista de los monitores (hardware) disponibles para realizar el experimento con sus respectivas resoluciones e identificadores. La pantalla principal tendrá el indicador 0.
Inputs	void.
Return	list.

get_available_monitors		
Descripción	Método estático. Entrega una lista de las configuraciones de monitor disponibles. Estas deben ser añadidas en la aplicación Monitor Center de PsychoPy y permiten setear características como la resolución de pantalla, tamaño físico de la misma, distancia del usuario, etc.	
Inputs	void.	
Return	list.	
load		
Descripción	Método que permite cargar en el objeto las configuraciones almacenadas en la base de datos.	
Inputs	name:	Nombre del perfil de configuración que se desea cargar.
Return	bool.	Verdadero si el proceso se completó satisfactoriamente, falso en caso contrario.
save		
Descripción	Método que permite guardar la configuración del objeto en la base de datos.	
Inputs	void.	
Return	bool.	Verdadero si el proceso se completó satisfactoriamente, falso en caso contrario.
copy		
Descripción	Método que permite realizar una copia de la configuración en otro objeto (copia profunda). Esto permite tener distintos perfiles de forma rápida variando pequeños elementos de cada uno.	
Inputs	name:	Nombre del nuevo perfil de configuración. Este no debe haber sido usado antes.
Return	objeto.	Devuelve una copia del objeto actual si se ingresa un nombre correctamente o None en caso contrario.
remove		
Descripción	Método que permite eliminar una configuración de la base de datos.	
Inputs	void.	
Return	bool.	Verdadero si la acción se realizó correctamente, falso en caso contrario.
get_iohub		
Descripción	En base a los atributos de la clase, correspondientes a la tabla master de la base de datos, esta función genera y retorna un diccionario con las configuraciones de los dispositivos requeridos por ioHub.	
Inputs	void.	

Return	dict.
get_configuration	
Descripción	Retorna un diccionario con los atributos contenidos en la clase. Estos datos se almacenarán en un archivo a modo de log para respaldar e indicar cual fue la configuración utilizada al ejecutar un experimento específico.
Inputs	void.
Return	dict.

Cuadro 1.4: Métodos implementados en la clase Master.

4. **ItemList:** Clase utilizada para manejar listas de objetos. Su implementación no considera el uso directo por parte de los usuarios del programa.

ItemList		
Descripción	Constructor. Inicializa los atributos de la clase y permite definir el tipo de datos que contendrá la lista.	
Inputs	itemclass.	Tipo de dato/clase de los objetos.
Return	void.	
item_add		
Descripción	Método protegido. Permite agregar un objeto a la lista siempre y cuando sea del tipo especificado en el contructor.	
Inputs	item:	Objeto a añadir.
Return	bool.	Verdadero si la acción se realizó correctamente, falso en caso contrario.
item_copy		
Descripción	Método protegido. Agrega una copia profunda del objeto seleccionado en el último lugar de la lista.	
Inputs	index:	Posición en la lista del objeto a copiar.
Return	bool.	Verdadero si el índice existe y se realiza la copia, falso en caso contrario.
item_delete		
Descripción	Método protegido. Elimina el objeto seleccionado.	
Inputs	index:	Posición en la lista del objeto a eliminar.
Return	bool.	Verdadero si el índice existe y es eliminado, falso en caso contrario.

item_move_up		
Descripción	Método protegido. Si es posible, mueve el objeto en un espacio hacia la parte superior de la lista (indices más pequeños).	
Inputs	index:	Posición en la lista del objeto a mover.
Return	bool.	Verdadero si la acción se realizó correctamente (fue posible mover el objeto), falso en caso contrario (no hay lugar o el índice no existe).
item_move_down		
Descripción	Método protegido. Si es posible, mueve el objeto en un espacio hacia la parte inferior de la lista (indices más altos).	
Inputs	index:	Posición en la lista del objeto a mover.
Return	bool.	Verdadero si la acción se realizó correctamente (fue posible mover el objeto), falso en caso contrario (no hay lugar o el índice no existe).
item_get_all		
Descripción	Método protegido. Retorna la lista de objetos.	
Inputs	void.	
Return	objeto.	list en caso de existir objetos, None en caso contrario.
item_get_by_index		
Descripción	Método protegido. Si es posible, devuelve el objeto asociado al índice.	
Inputs	index:	Posición en la lista del objeto seleccionado.
Return	objeto.	Objeto de tipo itemclass si el índice existe o None en caso contrario.
item_number		
Descripción	Método protegido. Devuelve el número de elementos de la lista.	
Inputs	void.	
Return	objeto.	int si existen objetos en la lista o None en caso contrario.

Cuadro 1.5: Métodos implementados en la clase ItemList.

5. **Component:** Clase utilizada para manejar la configuración de los elementos que conforman un cuadro.

Component		
Descripción	Constructor. Inicializa los atributos del objeto.	
Inputs	void.	
Return	void.	
get_list		
Descripción	Método estático. Retorna una lista con todos los componentes que forman parte de un cuadro específico.	
Inputs	db: exp: tes: fra:	Objeto de base de datos (SaccadeDB) mediante el cual se extraerá la información. Código del experimento al que pertenecen los componentes de interés. ID de la tarea a la cual pertenecen los componentes de interés. ID del cuadro al cual pertenecen los componentes de interés.
Return	objeto.	list si existen elementos en la base de datos, None en caso contrario.
encode_image		
Descripción	Método privado. En caso de que el componente corresponda a una imagen, esta función la codifica para almacenar los datos en la base de datos en un campo de tipo BLOB.	
Inputs	void.	
Return	objeto.	Unicode que contiene la imagen codificada en 'base64' en caso de existir imagen, None en caso contrario.
decode_image		
Descripción	Método privado. Permite decodificar una imagen almacenada en 'base64'.	
Inputs	encimg:	unicode que contiene la imagen codificada en 'base64'.
Return	objeto.	Imagen PIL en caso de lograr decodificar, None en caso contrario.
load		
Descripción	Método que permite cargar en el objeto las configuraciones almacenadas en la base de datos.	
Inputs	db: exp: tes: fra:	Objeto de base de datos (SaccadeDB) mediante el cual se extraerá la información. Código del experimento al que pertenece el componente. ID de la tarea a la cual pertenece el componente. ID del cuadro al cual pertenece el componente.

	com:	ID del componente.
Return	bool.	Verdadero si el proceso se completó satisfactoriamente, falso en caso contrario.
save		
Descripción	Método que permite guardar los datos de un componente en la base de datos.	
Inputs	db: exp: tes: fra: com:	Objeto de base de datos (SaccadeDB) mediante el cual se guardará la información. Código del experimento al que pertenecerá el componente. ID de la tarea a la cual pertenecerá el componente. ID del cuadro al cual pertenecerá el componente. ID que se le otorgará al componente.
Return	bool.	Verdadero si el proceso se completó satisfactoriamente, falso en caso contrario.
copy		
Descripción	Método que entrega una copia profunda del objeto.	
Inputs	void.	
Return	objeto.	Objeto de tipo Component con los mismos atributos.
get_execution		
Descripción	Método a ser utilizado durante la ejecución de un experimento. Retorna un objeto que puede ser presentado en el monitor mediante una ventana de PsychoPy.	
Inputs	win:	Objeto de tipo Window perteneciente a PsychoPy que permite mostrar los estímulos por pantalla.
Return	objeto.	Estímulo de la familia 'psychopy.visual'. Actualmente se encuentran implementadas 5 figuras (cruz, cuadrado, círculo, gaussiana y flechas) además de la posibilidad de incluir imágenes.
get_configuration		
Descripción	Retorna un diccionario con los atributos contenidos en la clase.	
Inputs	void:	
Return	dict.	

Cuadro 1.6: Métodos implementados en la clase Component.

6. **Frame:** Clase utilizada para configurar el comportamiento de un cuadro específico de una tarea. Ya que los cuadros contienen y muestran componentes, esta clase se define como hija de ItemList inicializada para objetos de tipo Component.

Frame		
Descripción	Constructor. Inicializa los atributos del objeto.	
Inputs	void.	
Return	void.	
get_list		
Descripción	Método estático. Retorna una lista con todos los cuadros que forman parte de una tarea específica.	
Inputs	db: exp: tes:	Objeto de base de datos (SaccadeDB) mediante el cual se extraerá la información. Código del experimento al que pertenecen los cuadros de interés. ID de la tarea a la cual pertenecen los cuadros de interés.
Return	objeto.	list si existen elementos en la base de datos, None en caso contrario.
load		
Descripción	Método que permite cargar el cuadro y sus componentes desde la base de datos. Para cargar los componentes se hace uso del método privado load_components.	
Inputs	db: exp: tes: fra:	Objeto de base de datos (SaccadeDB) mediante el cual se extraerá la información. Código del experimento al que pertenece el cuadro. ID de la tarea a la cual pertenece el cuadro. ID del cuadro.
Return	bool.	Verdadero si el proceso se completó satisfactoriamente, falso en caso contrario.
load_components		
Descripción	Método privado. Busca en la base de datos todos los componentes asociados al cuadro de interés, lo carga de forma iterativa y los agrega a la lista.	
Inputs	db: exp: tes: fra:	Objeto de base de datos (SaccadeDB) mediante el cual se extraerá la información. Código del experimento al que pertenece el cuadro. ID de la tarea a la cual pertenece el cuadro. ID del cuadro.
Return	void.	

save		
Descripción	Método que permite guardar los datos de un cuadro y sus componentes en la base de datos. Para guardar los componentes se hace uso del método privado save_components.	
Inputs	db: exp: tes: fra:	Objeto de base de datos (SaccadeDB) mediante el cual se guardará la información. Código del experimento al que pertenecerá el cuadro. ID de la tarea a la cual pertenecerá el cuadro. ID que se le otorgará al cuadro.
Return	bool.	Verdadero si el proceso se completó satisfactoriamente, falso en caso contrario.
save_components		
Descripción	Método privado. Guarda en la base de datos cada uno de los componentes de la lista.	
Inputs	db: exp: tes: fra:	Objeto de base de datos (SaccadeDB) mediante el cual se guardará la información. Código del experimento al que pertenece el cuadro. ID de la tarea a la cual pertenece el cuadro. ID del cuadro.
Return	void.	
copy		
Descripción	Método que entrega una copia profunda del objeto.	
Inputs	void.	
Return	objeto.	Objeto de tipo Frame con los mismos atributos.
get_execution		
Descripción	Método a ser utilizado durante la ejecución de un experimento. Retorna un diccionario que contiene las configuraciones del cuadro y una lista con sus componentes. Para esto se hace uso de forma iterativa de la función get_execution de cada componente que forma parte del cuadro.	
Inputs	win:	Objeto de tipo Window perteneciente a PsychoPy que permite mostrar los estímulos por pantalla.
Return	dict.	

get_configuration	
Descripción	Retorna un diccionario que contiene los atributos del cuadro y una lista con los atributos de sus componentes. Para esto se hace uso de forma iterativa de la función get_configuration de cada componente que forma parte del cuadro.
Inputs	void:
Return	dict.

Cuadro 1.7: Métodos implementados en la clase Frame.

7. **Test:** Clase utilizada para configurar una tarea específica en un experimento. Ya que las tareas se componen de cuadros, esta clase se define como hija de ItemList inicializada para objetos de tipo Frame.

Test		
Descripción	Constructor. Inicializa los atributos del objeto.	
Inputs	void.	
Return	void.	
get_list		
Descripción	Método estático. Retorna una lista con todas las tareas que forman parte de un experimento específico.	
Inputs	db: exp:	Objeto de base de datos (SaccadeDB) mediante el cual se extraerá la información. Código del experimento al cual pertenecen las tareas de interés.
Return	objeto.	list si existen elementos en la base de datos, None en caso contrario.
load		
Descripción	Método que permite cargar la tarea y sus cuadros desde la base de datos. Para cargar los cuadros se hace uso del método privado load_frames.	
Inputs	db: exp: tes:	Objeto de base de datos (SaccadeDB) mediante el cual se extraerá la información. Código del experimento al que pertenece la tarea de interés. ID de la tarea.
Return	bool.	Verdadero si el proceso se completó satisfactoriamente, falso en caso contrario.

load_frames	
Descripción	Método privado. Busca en la base de datos todos los cuadros asociados a la tarea de interés, los carga de forma iterativa y los agrega a la lista.
Inputs	db: Objeto de base de datos (SaccadeDB) mediante el cual se extraerá la información. exp: Código del experimento al que pertenece la tarea de interés. tes: ID de la tarea.
Return	void.
save	
Descripción	Método que permite guardar los datos de una tarea y sus cuadros en la base de datos. Para guardar los cuadros se hace uso del método privado save_frames.
Inputs	db: Objeto de base de datos (SaccadeDB) mediante el cual se guardará la información. exp: Código del experimento al que pertenecerá la tarea. tes: ID que se le otorgará a la tarea.
Return	bool. Verdadero si el proceso se completó satisfactoriamente, falso en caso contrario.
save_frames	
Descripción	Método privado. Guarda en la base de datos cada uno de los cuadros de la lista.
Inputs	db: Objeto de base de datos (SaccadeDB) mediante el cual se guardará la información. exp: Código del experimento al que pertenece la tarea. tes: ID de la tarea a la cual pertenecerá el cuadro.
Return	void.
copy	
Descripción	Método que entrega una copia profunda del objeto.
Inputs	void.
Return	objeto. Objeto de tipo Test con los mismos atributos.

get_execution		
Descripción	Método a ser utilizado durante la ejecución de un experimento. Retorna un diccionario que contiene las configuraciones de la tarea y una lista con sus cuadros. Para esto se hace uso de forma iterativa de la función get_execution de cada cuadro que forma parte de la tarea.	
Inputs	win:	Objeto de tipo Window perteneciente a PsychoPy que permite mostrar los estímulos por pantalla.
Return	dict.	
get_configuration		
Descripción	Retorna un diccionario que contiene los atributos de la tarea y una lista con los atributos de sus cuadros. Para esto se hace uso de forma iterativa de la función get_configuration de cada cuadro que forma parte de la tarea.	
Inputs	void:	
Return	dict.	

Cuadro 1.8: Métodos implementados en la clase Test.

8. **Experiment:** Clase utilizada para configurar un experimento específico. Ya que los experimentos se componen de tareas, esta clase se define como hija de ItemList inicializada para objetos de tipo Test. Para utilizar las funciones de carga/guardado es necesario configurar el objeto de base de datos mediante la función set_database.

Experiment		
Descripción	Constructor. Inicializa los atributos del objeto.	
Inputs	void.	
Return	void.	
get_list		
Descripción	Método estático. Retorna una lista con todas las tareas que forman parte de un experimento específico.	
Inputs	db:	Objeto de base de datos (SaccadeDB) mediante el cual se extraerá la información.
Return	objeto.	list si existen elementos en la base de datos, None en caso contrario.

load		
Descripción	Método que permite cargar el experimento y sus tareas desde la base de datos. Para cargar las tareas se hace uso del método privado load_tests.	
Inputs	code:	Código que identifica al experimento de interés.
Return	bool.	Verdadero si el proceso se completó satisfactoriamente, falso en caso contrario.
load_tests		
Descripción	Método privado. Busca en la base de datos todos las tareas asociados al experimento de interés, las carga de forma iterativa y los agrega a la lista.	
Inputs	void.	
Return	void.	
save		
Descripción	Método que permite guardar los datos de un experimento y sus tareas en la base de datos. Para guardar las tareas se hace uso del método privado save_tests.	
Inputs	void.	
Return	bool.	Verdadero si el proceso se completó satisfactoriamente, falso en caso contrario.
save_tests		
Descripción	Método privado. Guarda en la base de datos cada una de las tareas de la lista.	
Inputs	void.	
Return	void.	
copy		
Descripción	Método que permite realizar una copia de la configuración en otro objeto (copia profunda).	
Inputs	code:	Código del nuevo experimento. Debido a que este es un campo único, no puede ser igual a alguno ya almacenado en la base de datos.
	version:	Versión del nuevo experimento. no pueden existir dos experimentos con el mismo nombre y versión en la base de datos.
Return	objeto.	Objeto de tipo Experiment si ingresa correctamente el nuevo código y versión, None en caso contrario.

get_iohub	
Descripción	En base a los atributos de la clase, correspondientes a las tablas de la base de datos experiment y exp_dialog, esta función genera y retorna un diccionario con las configuraciones del experimento requeridas por ioHub.
Inputs	void.
Return	dict.
get_execution	
Descripción	Método a ser utilizado durante la ejecución de un experimento. Retorna un diccionario que contiene las configuraciones asociadas a la ejecución del experimento y una lista con sus tareas. Para esto se hace uso de forma iterativa de la función get_execution de cada tarea que forma parte del experimento.
Inputs	win: Objeto de tipo Window perteneciente a PsychoPy que permite mostrar los estímulos por pantalla.
Return	dict.
get_configuration	
Descripción	Retorna un diccionario que contiene los atributos del experimento y una lista con los atributos de sus tareas. Para esto se hace uso de forma iterativa de la función get_configuration de cada tarea que forma parte del experimento.
Inputs	void:
Return	dict.

Cuadro 1.9: Métodos implementados en la clase Experiment.

9. **ExperimentHandler:** Clase utilizada para efectuar las configuraciones previas que permiten ejecutar un experimento.

ExperimentHandler	
Descripción	Constructor. Inicializa los atributos del objeto.
Inputs	void.
Return	void.

load_experiment		
Descripción	Método que verifica la existencia del perfil de configuración (Master) y el experimento (Experiment) a ser utilizados. Si todo está correcto habilita la ejecución del experimento.	
Inputs	db: mas: exp:	Objeto de base de datos (SaccadeDB) mediante el cual se extraerá la información. Nombre dle perfil de configuración a utilizar. Código del experimento a utilizar.
Return	bool.	Verdadero si esta habilitada la ejecución, falso en caso contrario.
save_execution_parameters		
Descripción	Si el experimento se encuentra habilitado, este método verifica la existencia de las carpetas en donde se almacenarán los resultados y guarda en ellas respaldos de los archivos de configuración acompañados del timestamp en que fueron generados.	
Inputs	void:	
Return	bool.	Verdadero si el experimento esta habilitado y se guardaron los archivos, falso en caso contrario.
execute_experiment		
Descripción	Si el experimento se encuentra habilitado y los archivos de configuración están disponibles se ejecuta el experimento. Para esto, se inicializa una instancia de la clase ExperimentRuntime.	
Inputs	void.	
Return	bool.	Verdadero si el experimento pudo ser ejecutado, falso en caso contrario.

Cuadro 1.10: Métodos implementados en la clase ExperimentHandler.

10. **ExperimentRuntime:** Clase que ejecuta el experimento. Hereda de ioHubExperimentRuntime, clase de ioHub encargada de iniciar el servicio por el cual se manejan todos los dispositivos de hardware.

ExperimentRuntime		
Descripción	Constructor. Inicializa los atributos de la clase y permite entregar la configuración del experimento a la rutina.	
Inputs	exp_cfg_dict.	Diccionario que almacena el experimento a ejecutar y la ubicación de los archivos de configuración necesarios para correr el servicio.
Return	void.	

run	
Descripción	Método reimplementado de la clase padre y que tiene por función definir tanto la inicialización de los dispositivos como el script del experimento a ejecutar. Es importante destacar que este método NO se ejecuta directamente, para ello se utiliza la función específica ExperimentRuntime.start.
Inputs	*args: No utilizado.
Return	void.

Cuadro 1.11: Métodos implementados en la clase ExperimentRuntime.

1.2.3. Tercera parte: Interfaz gráfica

Pendiente.