
**ACTUALIZACIÓN Y MEJORA DEL SOFTWARE
DYNVA PARA MESURAR LA AGUDEZA
VISUAL DINÁMICA**

DIEGO DELGADO DÍAZ

Director/a: LUIS ANTONIO BELANCHE MUÑOZ (Departamento de
Ciencias de la Computación)

Codirector/a: MARC ARGILÉS

Titulación: Grado de Ingeniería Informática

Especialidad: Computación

Memoria del proyecto

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

17/05/2023

En lo onírico y en lo físico, Madre.

Abstract

This Bachelor's Thesis focuses on the development of a computer-assisted instrument for measuring dynamic visual acuity (DVA) in the field of optometry. The project is part of the Computer Engineering degree at the Faculty of Informatics in Barcelona (FIB). The aim of the project is to develop a reliable and valid application to objectively measure the Dynamic Visual Acuity using optotypes generated by a computer. The methodology used includes (but not limited to) an analysis of the previous version of the software, and the development of a new version of the software.

Resum

Aquest TFG se centra en el desenvolupament d'un instrument assistit per ordinador per a mesurar l'agudesia visual dinàmica (DVA) en el camp de l'optometria. El projecte forma part del grau d'Enginyeria Informàtica impartit per la Facultat d'Informàtica de Barcelona (FIB). L'objectiu d'aquest projecte és desenvolupar una aplicació fiable i vàlida per a mesurar de manera objectiva l'agudesia visual dinàmica utilitzant optotips generats per ordinador. La metodologia inclou, però no es limita, a una anàlisi de la versió anterior del programari i el desenvolupament d'una nova versió d'aquest programari.

Resumen

Este TFG se centra en el desarrollo de un instrumento asistido por ordenador para medir la agudeza visual dinámica (DVA) en el campo de la optometría. El proyecto forma parte del grado de Ingeniería Informática impartido por la Facultat de Informàtica de Barcelona (FIB). El objetivo de este proyecto es desarrollar una aplicación fiable y válida para medir de forma objetiva la agudeza visual dinámica utilizando optotipos generados por ordenador. La metodología incluye, pero no se limita, a un análisis de la versión anterior del software y el desarrollo de una nueva versión de este programa.

Índice de contenido

Índice de figuras	4
Índice de tablas	4
1. Introducción y contextualización	6
1.1. Contexto	6
1.2. Conceptos	7
1.2.1. Test <i>DVA</i> : (DynVA 3.0)	7
1.3. Identificación del problema	8
1.4. Agentes implicados	8
2. Justificación	8
2.1. Estudio soluciones existentes	8
3. Alcance	9
3.1. Objetivos	9
3.2. Requerimientos	9
3.2.1. Requerimientos funcionales	9
3.2.2. Requerimientos no funcionales	9
4. Metodología	10
4.1. Herramientas	10
4.1.1. Software	10
4.1.2. Hardware	11
5. Planificación temporal	12
5.1. Descripción de las tareas	12
5.2. Recursos	14
5.2.1. Recursos humanos	14
5.2.2. Recursos materiales	15
6. Gestión del riesgo	15
7. Gestión económica	17
7.1. Presupuesto	17
7.1.1. Costes de personal	17
7.1.2. Costes genéricos	18
7.1.3. Contingencia	19
7.1.4. Imprevistos	19
7.1.5. Coste total	20
7.2. Control de gestión	21

8. Sostenibilidad	21
8.1. Autoevaluación	21
8.2. Dimensión Económica	21
8.3. Dimensión Ambiental	22
8.4. Dimensión Social	22
9. Desarrollo del software	23
9.1. Definición de los requisitos y especificaciones del software	23
9.1.1. Funcionalidades del software	23
9.2. Arquitectura del software	24
9.2.1. Lógica de la aplicación	24
9.2.2. Interfaz del usuario	27
9.2.3. Gestión de datos	31
9.3. Implementación del software	32
9.3.1. Animación del optotipo	32
9.3.2. Aplicación de la configuración	33
9.4. Fase de pruebas y errores	36
9.5. Documentación para el usuario	36
10. Conclusión	37
11. Bibliografía	38
12. Anexos	39
Anexo A. Diagrama de Gantt	39
Anexo B. Manual del usuario	40

Índice de figuras

1. Test Snellen, utilizado actualmente para determinar la agudeza visual[2].	6
2. Optotipo anillo disco Palomar[5], orientación norte.	7
3. Panel de control antiguo <i>Dyn VA 3.0</i>	8
4. Diagrama realizado con la herramienta <i>draw.io</i> . [16]	25
5. Captura de pantalla de guardado.	26
6. Captura de pantalla, mostrando las partes de un test.	27
7. Captura de la pantalla de inicio.	28
8. Capturas de las pantallas de guardado de la configuración.	31
9. Captura de la pantalla de guardado de los resultados.	31
10. Configuración del número de tests y valores iniciales.	34

Índice de tablas

1. Requerimientos a nivel hardware de <i>Godot</i> [11].	11
--	----

2.	Tabla de tareas con los recursos necesarios.	15
3.	Costes de personal a partir de la guía de mercado laboral de <i>Hays</i> [14]. . .	17
4.	Tabla de partidas por tarea.	18
5.	Costes genéricos.	19
6.	Tabla de contingencia del 15% por tipo de gasto.	19
7.	Tabla de sobrecostes añadido por imprevistos.	20
8.	Tabla del presupuesto final del proyecto.	20

1. Introducción y contextualización

La optometría es una disciplina sanitaria, pero no médica, que se encarga del estudio del sistema visual. Las personas que practican un deporte, especialmente a nivel de competición, deben estar en perfectas condiciones físicas para que el cuerpo responda a los estímulos externos de la manera más rápida y precisa posible, y una **buena visión es fundamental** para ello. La optometría deportiva trabaja de manera individual con cada persona detectando sus problemas y las necesidades de desarrollo visual que precisa según el deporte que practica y su rol en él.

Hasta hace relativamente poco tiempo no se daba demasiada importancia al cuidado de la visión para mejorar el rendimiento deportivo. Sin embargo, lograr una buena visión puede marcar la diferencia entre ser un buen deportista o ser uno mejor [1].

La agudeza visual es la capacidad del sistema de visión para percibir, detectar o identificar objetos especiales con unas condiciones de iluminación buenas. Para una distancia al objeto constante, si el paciente ve nítidamente una letra pequeña, tiene más agudeza visual que otro que no la ve.

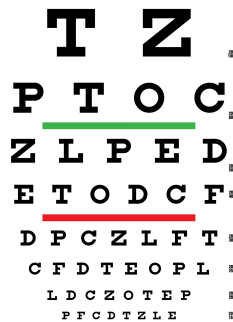


Figura 1: *Test Snellen, utilizado actualmente para determinar la agudeza visual[2].*

La **agudeza visual dinámica** se define como la habilidad de discriminar detalles de un objeto cuando existe movimiento relativo entre dicho objeto y el observador [3].

Este trabajo de fin de grado, pretende desarrollar una aplicación para visualizar una serie de optotipos generados por ordenador con la finalidad de determinar la agudeza visual dinámica de forma objetiva, válida y fiable.

1.1. Contexto

Dentro del Grado en Ingeniería Informática, impartido por la Facultad de Informática de Barcelona, existen diferentes menciones. Este trabajo de fin de grado pertenece a la mención de computación, concretamente, al ámbito de los gráficos por ordenador o computación gráfica.

El proyecto parte del artículo de la Dra. Quevedo[4], donde se diseñó y posteriormente se analizó cualitativamente la validez y la confiabilidad de un nuevo instrumento asistido

por computadora (DynVA 3.0) para la medición de la agudeza visual dinámica (*DVA*¹). También, de forma paralela, este proyecto parte de la base de una mejora de *software* que se realizó a inicios del 2017. En esta última versión se utilizaba una tecnología más acorde a la época respecto la anterior versión, pero se dejó a medias por causas desconocidas. En este proyecto se pretende dar una imagen nueva al *software*, haciendo que su uso funcional sea el adecuado.

1.2. Conceptos

A continuación, se definen los conceptos/términos clave de este proyecto, así como sus temas relacionados.

1.2.1. Test *DVA*: (DynVA 3.0)

Este test, parte de la siguiente procedimiento: *DVA* se mide de forma binocular² instruyendo a participantes a indicar la orientación percibida de optotipo anillo disco Palomar (Figura 2) con el teclado numérico[4].

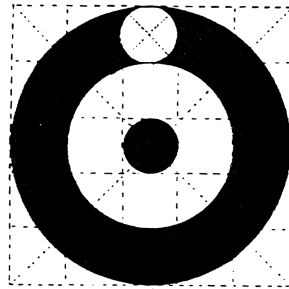


Figura 2: *Optotipo anillo disco Palomar*[5], *orientación norte*.

El test se puede dividir en dos sub tests básicamente, pero parten de la misma premisa, el optotipo se mueve de un lado a otro, en línea recta y la orientación de la figura es al azar entre las 8 disponibles. Los nombres y diferencias de cada subtest son:

- **Speed series:** El optotipo mantiene la velocidad, y a cada intervalo de tiempo t , el **optotipo cambia de tamaño**.
- **Size series:** El optotipo mantiene la velocidad, y a cada intervalo de tiempo t , el **optotipo cambia de velocidad**.

¹*DVA: Dynamic Visual Acuity.*

²Tipo de visión en que los dos ojos se utilizan conjuntamente.

1.3. Identificación del problema

En este trabajo se va desarrollar una solución que satisfaga la premisa de poder generar y mejorar el test de Agudeza Visual Dinámica (*Dyn VA 3.0*) de forma correcta y eficiente. El problema recae básicamente en decidir si se va utilizar la solución del 2017 y corregir los errores de *software* que existen o diseñar la aplicación desde cero.

1.4. Agentes implicados

El sistema a desarrollar está dirigido pero no limitado a deportistas de élite que quieran desarrollar una mejor visión para marcar la diferencia a nivel competitivo. Dicho sistema también puede dirigirse a personas mayores, que por normal general son las más afectadas por trastornos de visión. pero cualquier otra persona fuera de esos subgrupos de la población también se puede beneficiar de esta aplicación.

2. Justificación

La agudeza visual dinámica es una capacidad cuya medición, aun en la actualidad, no suele formar parte de las baterías de test de los centros optométricos, y disponer de aplicaciones de *software* con tecnología actual para poder ayudar a diagnosticar, tratar y mejorar dicha capacidad se ha convertido en una necesidad. En consecuencia, el problema planteado en la sección 1.3, se planteará una serie de argumentos porque es más conveniente desarrollar una nueva aplicación en vez de reutilizar el código existente. También se planteará que lenguaje/tecnología se utilizarán para desarrollar la aplicación.

2.1. Estudio soluciones existentes

La actual aplicación está desarrollada en *Visual Studio*³ (se desconoce la versión) y esta tecnología está diseñada para aplicaciones de escritorio. Es cierto que se pueden mostrar animaciones generadas por animación, pero el lenguaje no permite abstraerse tanto como otras tecnologías actuales por lo tanto se hace más difícil la lectura y comprensión. De forma alternativa, puede parecer más sugerente y sencillo reutilizar el código fuente, pero no hay ninguna garantía de que sea más fácil corregirlo que empezar de nuevo, así que esta incertidumbre se convierte en el segundo inconveniente.

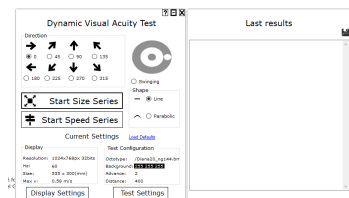


Figura 3: Panel de control antiguo *Dyn VA 3.0*.

³Microsoft *Visual Studio* es un entorno de desarrollo integrado para *Windows* y *macOS*.

3. Alcance

Como en todo proyecto, es importante definir el alcance de este, pues el tiempo de desarrollo es limitado y es necesario definir los objetivos y requerimientos de la aplicación. A continuación, se definen los objetivos, los requerimientos no funcionales y los obstáculos y riesgos del proyecto.

3.1. Objetivos

El objetivo principal de este trabajo es el desarrollo, implementación de los diferentes tipos de test requeridos para evaluar la agudeza visual dinámicos con unos parámetros que variaran como la velocidad/tamaño del test en función de la distancia del usuario a la pantalla o el usuario. Seguidamente, se divide el objetivo en los siguientes subobjetivos:

1. Generar una aplicación que permita generar unos tests visuales desde una interfaz gráfica.
2. Ser capaz de poder producir test visuales a partir de unos parámetros definidos por el usuario.
3. Poder generar una puntuación que mida la agudeza visual dinámica a través de cuántos aciertos ha obtenido el usuario mediante el teclado numérico.

3.2. Requerimientos

3.2.1. Requerimientos funcionales

A pesar de que los principales requerimientos funcionales del sistema se definen en el objetivo, a continuación, se definen otros requisitos necesarios para el funcionamiento de la aplicación;

1. Compatibilidad. Se plantea que la aplicación sea compatible con diferentes tipos de pantalla, tanto en tamaño, resolución o frecuencia de refresco. También se plantea que sea compatible con el mínimo de requerimientos a nivel *hardware* de *Godot* reflejado en la Figura 1.
2. Mantener las funcionalidades presentes en la antigua versión de *software* de *DynVA 3.0*.

3.2.2. Requerimientos no funcionales

A continuación se definen los requisitos no funcionales del sistema, es decir, aquellos requerimientos que no hablan directamente del funcionamiento del sistema, sin embargo, se han de tener en cuenta desde el principio para el desarrollo de la aplicación:

1. Usabilidad. La aplicación ha de ser sencilla de utilizar tanto para el examinador que configura los tests, como para el usuario que quiere ser examinado. Esto requerirá crear una interfaz gráfica de fácil uso.
2. Eficiencia. La aplicación se tiene que poder ejecutar con el mínimo nivel de hardware, por lo tanto se tiene que tener en mente no hacer operaciones computacionales elevadas y no crear código innecesario.
3. Eficacia. La aplicación no debe fallar bajo ningún concepto, debe hacer su tarea lo mejor posible y no dar ningún tipo de error. Esto último puede ser trivial, pero no lo es.

4. Metodología

La metodología ágil de desarrollo de *software* cumple con los requisitos expuestos anteriormente, la idea es definir ciclos de desarrollo cortos donde se diseñe, implemente una funcionalidad.

Concretamente, siguiendo con la metodología ágil y se realizarán reuniones semanales con las personas implicadas en este proyecto, en las que se comprobarán los objetivos propuestos, y en función del estado, se realizarán modificaciones en la planificación y/o se establecerán las nuevas tareas a realizar.

4.1. Herramientas

Para facilitar el seguimiento de los objetivos se usará *Trello*[6], se trata de una herramienta *online* que permite establecer tareas en forma de tarjetas, lista y tableros virtuales. El objetivo es usar un tablero para el proyecto donde una lista representa el grupo de tareas a realizar en una semana, y cada tarjeta es una tarea a realizar.

El control de versiones es una herramienta en cualquier proyecto informático, permite hacer un seguimiento de los cambios del proyecto y sirve como copia de seguridad. Para ello se usará *Github*[7].

Por último, se utilizará la herramienta *Gantter*[8] para la planificación de las tareas. A través del diagrama de *Gantt*, se realizará un control del tiempo dedicado a cada tarea para asegurar la finalización del proyecto en el plazo establecido. En las reuniones semanales, se validará el tiempo gastado en cada tarea y se actualizará el diagrama en consecuencia.

Para el desarrollo de la aplicación es necesario un conjunto de herramientas *software* y *hardware*.

4.1.1. Software

Un *game engine* es un *software* con una interfaz bien definida que permite el desarrollo de videojuegos. Hay varias empresas que se dedican a desarrollar este tipo de

soluciones, quizá la más conocida es *Unity*[9] que no es *open source*⁴ pero sí es gratuita mientras no se tengan ganancias de más de cien mil dólares por año. *Unity* se desarrolló en el 2005 y se ha utilizada para crear un gran número de videojuegos muy conocidos alrededor del mundo.

Otro *game engine* aparte de *Unity*, sería *Godot*[10] el cuál sí es *open source*. Se estrenó en el 2014 y poco a poco se va haciendo hueco entre sus competidores. Esta tecnología permite hacer *scripts* con un lenguaje propio llamado *GDScript* basado en *Python* y en *Lua*. *Godot* es una herramienta sencilla de utilizar pero también tiene sus inconvenientes: no es ideal para juegos con gráficos complejos en 3D. Este último inconveniente no es un problema ya que las animaciones generadas por el test se desarrollan en un mundo 2D.

Dado que el autor de este TFG es un interesado del mundo *open source* y tiene conocimientos avanzados del lenguaje *Python*, se escoge utilizar *Godot* como herramienta para desarrollar la aplicación.

4.1.2. Hardware

La aplicación será ejecutada en un ordenador, los requerimientos vienen precedidos por la tecnología que escojamos, que en este caso es *Godot*:

CPU	2 cores
OS:	<i>Windows 7</i> mínimo, <i>macOS 10.10</i> mínimo, Linux (64-bit or 32-bit x86)
RAM:	4GB
Tarjeta gráfica	Soporte de <i>OpenGL 3.3 Core Profile</i>

Tabla 1: *Requerimientos a nivel hardware de Godot[11].*

Por otro lado, para mostrar los test visuales es necesario una pantalla. También, hará falta un teclado numérico, el cual el usuario utilizará para escoger la dirección de optotipo del test.

⁴*Open source: Software* de código abierto es el *software* cuyo código fuente y otros derechos que normalmente son exclusivos para quienes poseen los derechos de autor, son publicados bajo una licencia de código abierto o forman parte del dominio público.

5. Planificación temporal

Con el objetivo de finalizar este trabajo de fin de grado en la fecha estimada y cumplir con los objetivos planteados previamente, se realiza una planificación temporal del proyecto dividido en tareas.

El trabajo empieza el día 5 de septiembre de 2022 y finaliza el 17 de mayo de 2023. En total, el desarrollo del proyecto se llevará a cabo a lo largo de 109 días aproximadamente y unas 327 horas aproximadamente. La dedicación diaria de media será de 1.5 horas aproximadamente. Cada dos semanas semana se quedará con los responsables según disponibilidad horaria y se mostrarán los avances y las mejoras que requiere el proyecto.

En la sección 12 Anexos se muestra el se muestra el diagrama *Gantt* de las tareas (Anexo A).

5.1. Descripción de las tareas

En esta sección se detallan las tareas a realizar de forma individual, pero se agrupan por bloques para distinguir con mayor facilidad las distintas fases del proyecto.

GP - Gestión del proyecto

La gestión del proyecto es esencial para planificar, definir y documentar el trabajo a realizar, además, engloba las reuniones para la validación y propuesta de objetivos semanales. Se estima que en global el grupo de gestión tendrá una carga de 115 horas.

GP.1 - Alcance

Antes de empezar con el proyecto es necesario acotar el desarrollo, por este motivo, se dedica tiempo inicial a definir qué se quiere conseguir con el trabajo, qué se va a desarrollar y qué medios serán necesarios. La duración ha sido de 5 horas.

GP.2 - Planificación

Para cumplir con los objetivos propuestos en la definición del alcance del proyecto, se realiza una planificación temporal, así como de recursos y requerimientos asociados a cada tarea. Además, se definen los riesgos y obstáculos, y se plantean tareas alternativas para solventarlos. La duración de esta fase ha sido de 10 horas.

GP.3 - Presupuesto

Se realizará un presupuesto para cuantificar el coste del proyecto, para ello, se realizarán partidas por cada tarea teniendo en cuenta los costes de personal y equipo, además, se cuantificarán los costes genéricos y partidas de imprevistos. Se estima una dedicación de 5 horas.

GP.4 - Informe de sostenibilidad

Se analizará a partir de un informe el impacto medioambiental, económico y social del proyecto, en concreto, de las fases de planificación y desarrollo. El tiempo estimado para realizar el informe es de 5 horas.

GP.5 - Reuniones

Como en cualquier proyecto de desarrollo de aplicaciones, es necesario realizar reuniones frecuentemente para analizar los resultados obtenidos según los objetivos semanales, y decidir si se han de cambiar aspectos de la planificación. Se prevén reuniones semanales de 1 hora con los directores del proyecto. En total se estima una duración de 20 horas.

GP.6 - Documentación

Una parte importante del TFG es la memoria final, por lo tanto, a lo largo del desarrollo del proyecto se han de documentar las distintas fases. La documentación se realizará de forma paralela al resto del proyecto, de esta forma se irán incorporando las secciones en las que se trabaje. La duración estimada es de 60 horas.

GP.7 - Presentación

Por último, una vez finalizada la documentación, es necesario preparar la presentación para el tribunal que evaluará el TFG. Se preparará material de soporte para la presentación, así como el guión, y se realizarán ensayos. En total la duración estimada es de 10 horas.

TP - Trabajo previo

En este apartado se especifican las tareas a realizar antes del desarrollo del trabajo, es decir, tareas de preparación y estudio previo. Puesto que se trata de una fase de preparación, se puede realizar paralelamente a la mayoría de las tareas de gestión del proyecto. Se estima una duración de 50 horas.

TP.1 - Aprendizaje

Obtención de los conocimientos necesarios sobre las herramientas y lenguajes involucrados. Aunque ya se posean conocimientos sobre lenguajes de programación, hace falta entender con profundidad:

1. *GDScript*: comprender lenguaje que se utiliza en *Godot*.
2. *Godot*: comprender interfaz que propone dicha herramienta así cómo hacer uso de las técnicas de visionado y animación de objetos.

Con tal de obtener una buena base en estos conceptos se prevé una dedicación de 50 horas al aprendizaje.

DA - Desarrollo aplicación**DA.1 - Diseño**

Habiendo decidido las funcionalidades a crear para la aplicación teniendo en cuenta los objetivos establecidos, se para al diseño de estas funcionalidades. Una vez listo, se validará el diseño para que se cumplan los objetivos y garantizar la viabilidad de la solución. Se estiman una duración de 12 horas.

DA.2 - Implementación

La implementación del producto es traducir el diseño de las funcionalidades a código. La programación es la parte más importante y extensa del proyecto. A grandes rasgos se pueden diferenciar la parte gráfica y la mecánica que va por detrás. Es la parte con más peso, con un volumen estimado de 150 horas.

5.2. Recursos**5.2.1. Recursos humanos**

En este proyectos se encuentran tres roles diferentes: jefe de proyecto, programador y *tester*. No obstante, teniendo en cuenta que este TFG se realiza por una persona, será el autor el encargado de asumir los diferentes roles en función de la tarea a realizar.

1. Jefe de proyecto: Se encarga de la planificación del proyecto, liderar las reuniones con el equipo y escribir la documentación.
2. Programador: Es la persona encargada de implementar el sistema.
3. *Tester*: Se ocupa de realizar las pruebas de validez del sistema, debe diseñar las pruebas, ejecutarlas y presentar un informe para poder arreglar los errores encontrados.

5.2.2. Recursos materiales

A continuación, se exponen los recursos materiales necesarios para la realización del proyecto.

1. Ordenador para realizar las tareas de gestión y producción del proyecto.
2. *Overleaf*[12]: Entorno de desarrollo para crear documentos con \LaTeX ⁵.
3. *Godot*: Motor gráfico que servirá como entorno de desarrollo para la aplicación.
4. *Ganttter*[8]: Herramienta *web* para la creación de diagramas de Gantt.

A continuación se expone la Tabla 2 la asignación de recursos materiales a cada tarea.

Id.	Tareas	Duración	Recursos materiales	Recursos humanos
GP	Gestión del proyecto	115h	-	-
GP.1	Alcance	5h	Ordenador	JP
GP.2	Planificación	10h	Ordenador	JP
GP.3	Presupuesto	5h	Ordenador	JP
GP.4	Informe de sostenibilidad	5h	Ordenador	JP
GP.5	Reuniones	20h	Ordenador	JP,T,P
GP.6	Documentación	60h	Ordenador	JP
GP.7	Presentación	10h	Ordenador	JP
TP	Trabajo previo	50h	-	-
TP.1	Aprendizaje	50h	Ordenador	P
DA	Desarrollo aplicación	162h	-	-
DA.1	Diseño	12h	Ordenador	P
DA.2	Implementación	150h	Ordenador	P,T
-	Total	327h	-	-

JP: Jefe de Proyecto, T: *Tester*, P: Programador.

Tabla 2: Tabla de tareas con los recursos necesarios.

6. Gestión del riesgo

Es importante prever los riesgos y obstáculos que pueden surgir durante el desarrollo del TFG. Todo proyecto tienen diferentes dificultades que se deben analizar anticipadamente para minimizar su impacto. Es importante prever los riesgos y obstáculos que pueden surgir durante el desarrollo. A continuación se describen los posibles obstáculos.

1. Dado que el proyecto anterior tiene errores y algunas funcionalidades quedan por añadir, es importante tener una buena comunicación con los solicitantes de este aplicación, no tener una buena comunicación conllevaría un aumento de tiempo al desarrollo.

⁵ \LaTeX : es un sistema de composición de textos, orientado especialmente a la creación de libros, documentos científicos y técnicos que contengan fórmulas matemáticas[13].

2. Cuestiones de código. Un riesgo importante que conlleva el desarrollo de software es la mala calidad de código, se puede mitigar dicho riesgo con las siguientes acciones.
 - Probar el código con frecuencia.
 - Resolver fallos y errores lógicos cuando se encuentran.
 - Utilizar las mejores prácticas de código.
3. Errores en *Godot*: A pesar de que dicha herramienta es utilizada por miles de usuarios, todavía es reciente y pueden contener *bugs*. Si aparece un error que bloquee el desarrollo del proyecto, se tendrá que añadir una tarea a la planificación para desarrollar una solución alternativa a la proporcionada por la librería. Se estima que podría añadir entre 20 y 40 horas de desarrollo.
4. Rendimiento: la aplicación tiene que ser suficiente optimizada para que se pueda ejecutar en equipos con un *hardware* relativamente bajo en especificaciones.

7. Gestión económica

Una vez realizada la planificación temporal del proyecto, se van a estimar los costes necesarios para el desarrollo. Se identifican diferentes tipos de costes asociados al personal, espacio de trabajo, y a las herramientas y dispositivos usados. Además, para superar los obstáculos que aparezcan y asumir los costes no programados, se realiza un plan de contingencia, una partida de imprevistos y se exponen mecanismos para controlar el presupuesto.

7.1. Presupuesto

7.1.1. Costes de personal

A partir de la planificación por tareas se calcula el coste de personal, se tienen en cuenta los 3 roles definidos anteriormente: jefe de proyecto, programador y *tester*. En la Tabla 3 se ve el coste por hora de cada puesto, los datos han sido obtenidos de la empresa de reclutamiento *Hays*:

Rol	Coste por hora
Jefe de proyecto	30€/h
Programador	16€/h
<i>Tester</i>	16€/h

Tabla 3: Costes de personal a partir de la guía de mercado laboral de *Hays*[14].

En la Tabla 4 se detallan las partidas por tarea a partir de los costes de personal de la Tabla 3, y se estima el coste de la seguridad social multiplicando el coste por 1,3. En total el coste del personal del proyecto es de 12.847€.

Id.	Tareas	Duración	Roles	Coste	Coste SS
GP	Gestión del proyecto	115h	-	4.090€	5.317€
GP.1	Alcance	5h	JP	150€	195€
GP.2	Planificación	10h	JP	300€	390€
GP.3	Presupuesto	5h	JP	150€	195€
GP.4	Informe de sostenibilidad	5h	JP	150€	195€
GP.5	Reuniones	20h	JP,T,P	1.240€	1.612€
GP.6	Documentación	60h	JP	1.800€	2.340€
GP.7	Presentación	10h	JP	300€	390€
TP	Trabajo previo	50h	-	800€	1.040€
TP.1	Aprendizaje	50h	P	800€	1.040€
DA	Desarrollo aplicación	162h	-	4.992€	6.490€
DA.1	Diseño	12h	P	192€	250€
DA.2	Implementación	150h	P,T	4.800€	6.240€
-	Total	327h	-	9.882€	12.847€

JP: Jefe de Proyecto, T: *Tester*, P: Programador.

Tabla 4: Tabla de partidas por tarea.

7.1.2. Costes genéricos

A continuación se describen todos los costes independientes a las tareas del proyecto:

- **Espacio físico:** para realizar presencialmente las diversas tareas del proyecto. Se realiza desde el despacho de una vivienda ubicada en Barcelona. El proyecto se desarrolla a lo largo de 4 meses, teniendo en cuenta que cada mes son 90€, el coste final es de 360€.
- **Ordenador:** ya que este proyecto se realiza por una única persona solo es necesario una unidad de ordenador. El ordenador tiene un hardware muy por encima de la media y también sus periféricos.
- **Software:** se utiliza el mismo que se ha mencionado en los anteriores apartados. Todos tienen la característica que son *Open Source* menos *Gantter* que tiene un coste de 5€ al mes, aunque el primer mes es gratuito y se plantea solo utilizarlo únicamente en ese término, así pues todas las herramientas tienen coste 0.

La tabla 5 muestra los costes genéricos identificados anteriormente, su amortización y vida útil. Para calcular las amortizaciones, se ha calculado el coste por hora teniendo en cuenta que un año tiene 220 días hábiles y 8 horas laborables al día:

$$\text{Amortización} = \text{CosteDispositivo} / (\text{VidaÚtil} * 220 * 8) * \text{horas}$$

Para el despacho no hace falta calcular el coste de amortización ya que es un activo alquilado, el resto de activos, al tener un coste inicial 0, su correspondiente amortización acaba valorándose en 0€.

Elemento	Coste	Vida útil	Coste amortizado
Despacho	90€/mes	-	-
Ordenador	2.500€	5 años	92,89€
Software	0€	0	0€
Total	2.860€/h	-	93€

Tabla 5: *Costes genéricos.*

7.1.3. Contingencia

Como en todo proyecto, es importante añadir un sobrecoste para cubrir obstáculos e imprevistos. En este caso se ha decidido fijar un 15% de sobrecoste. En la Tabla 6 se detalla la contingencia total del proyecto:

Tipo	Coste	Contingencia
Espacio	360€	54€
Hardware	92,89€	13,93€
Personal	12.847€	1.927,05€
Total	13.300€	1.954€

Tabla 6: *Tabla de contingencia del 15% por tipo de gasto.*

7.1.4. Imprevistos

Por último, se calcula el coste de los obstáculos que puedan surgir durante el desarrollo del proyecto. Los imprevistos se presentan en la planificación temporal, a continuación, sólo se cuantifica el riesgo y el coste que pueden causar, en la Tabla 7 se detalla el coste.

- Fallo de dispositivo. En caso de que el ordenador falle será necesario comprar uno nuevo. Se estima un 1% de riesgo ya que el ordenador y sus componentes son nuevos.
- Aumento tiempo de desarrollo, se añadirán 20 horas de desarrollo a la aplicación y 2 horas de *testing*. El coste total sería de 20 horas de programador y 2 horas de *tester*, por lo tanto, 352€, se cuantifica el riesgo en un 5% ya que se hacen reuniones de forma periódica con los solicitantes de la aplicación.
- Errores en *Godot*: Tal y como se describe en la planificación temporal, en caso de que hubiera algún *bug*, se habría que buscar otra solución alternativa, por lo tanto se añadirían 25 horas de trabajo y 3 horas de *tester*, en total 448€. Se estima un riesgo relativamente bajo ya que dicha herramienta es utilizada por miles de usuarios.

Imprevisto	Coste	Riesgo	Coste total
Ordenador	2.500€	1 %	25€
Aumento tiempo desarrollo	352€	5 %	16,25€
Errores en Godot	448€	5 %	22,4€
Total	3.300€	-	63,65€

Tabla 7: *Tabla de sobrecostes añadido por imprevistos.*

7.1.5. Coste total

Una vez presentados todos los costes del proyecto, en la Tabla 8 se presenta el presupuesto final del trabajo. El coste total del proyecto es de 15.459€.

Tipo	Coste
Personal	12.847€
Espacio	360€
Hardware	92,89€
Contingencia	1.994,98€
Imprevistos	63,65€
Total	15.359€

Tabla 8: *Tabla del presupuesto final del proyecto.*

7.2. Control de gestión

Una vez definido el presupuesto inicial, se definen los mecanismos de control necesarios para evitar desviaciones, así como indicadores numéricos que ayuden al control. En las reuniones semanales, cada vez que se acabe una tarea, se actualizará el presupuesto con las horas reales y se comparará con las horas estimadas.

Para controlar los imprevistos, al finalizar una tarea en *Trello* también se apuntarán los gastos extra que se hayan producido en una hoja de *Excel*, y se compararán con la previsión de imprevistos y contingencia. De esta forma, rápidamente se podrá detectar cualquier desviación y predecir si es necesario recortar alguna tarea o aumentar el presupuesto.

A continuación se presenta los descriptores numéricos para el control:

- Desviación coste personal por tarea:
(coste_estimado - coste_real) * horas_reales
- Desviación realización tareas:
(horas_estimadas - horas_reales) * coste_real
- Desviación total en la realización de las tareas:
(coste_estimado_total - coste_real_total)
- Desviación total de recursos (*hardware*, personal, espacio):
(coste_estimado_total - coste_real_total)
- Desviación total coste de imprevistos:
(coste_estimado_imprevistos - coste_real_imprevistos)
- Desviación total de horas:
(horas_estimadas - horas_reales)

8. Sostenibilidad

8.1. Autoevaluación

Después de realizar la encuesta de sostenibilidad he visto que mis conocimientos relacionados con la materia son escasos. Algunas preguntas me han hecho dar cuenta de que los efectos colaterales de una solución relacionada con el mundo de la informática puede ser muy drásticos. También me he dado cuenta que el *hardware* que se ejecuta los programas requiere mucha energía y recursos para ser producido.

Otro aspecto importante es el social. En este ámbito si que personalmente conozco las ventajas de las herramientas colaborativas, y también que el *software* ha de ser ético, transparente y accesible, ya que su objetivo es ayudar a la gente.

8.2. Dimensión Económica

El coste estimado de este proyecto incluye únicamente los recursos necesarios para el correcto desarrollo y funcionamiento de la aplicación y también se han estudiado los posibles riesgos que hacen variar este presupuesto, por lo tanto se intenta no hacer un mal uso de dinero ni recursos.

8.3. Dimensión Ambiental

Respecto a la sostenibilidad ambiental, se ha tenido en cuenta de que el *software* no haga uso de recursos innecesarios o ineficientes, aunque está el impacto negativo latente a raíz de la electricidad consumida. No obstante, el proyecto ha sido desarrollado en un despacho ubicado en mi vivienda, así que no se provocará ningún impacto medioambiental por desplazamientos. A parte de la electricidad, no hay otro tipo de recurso que impacte negativamente el medio ambiente, ya que no hace falta ningún *hardware*, se puede desarrollar y utilizar con equipos ya amortizados y no se produce CO₂ durante la elaboración del proyecto. Se ha hecho hincapié en utilizar herramientas *Open Source* con tal de aprovechar la energía que se gastaría desarrollando funcionalidades ya existentes.

8.4. Dimensión Social

Personalmente, este proyecto puede enriquecerme de varias formas, no solamente con el conocimiento sobre el desarrollo de *software* y cómo aplicar las metodologías ágiles. También se aprenderá a cómo llevar sesiones de *testing* con los solicitantes de este proyecto para poder revisar funcionalidades y ver si se pueden replantear con el fin de que sean más intuitivas y utilizables. Y por último, y no por ello menos importante, este proyecto nace de la necesidad del mundo sanitario para poder diagnosticar y tratar la salud visual de las personas, y sobre todo de las personas de edad avanzada que forman parte del grupo de riesgo de la sociedad actual.

9. Desarrollo del software

En esta sección se describirán todas las etapas del proceso de desarrollo:

- Definición de los requisitos y especificaciones del software.
- Descripción de la arquitectura del software y la explicación de cómo se ha llevado a cabo su diseño.
- Implementación del software.
- Realización de pruebas de software y corrección de errores y problemas que se hayan encontrado.

9.1. Definición de los requisitos y especificaciones del software

En este apartado se detallan los requerimientos que el software debe cumplir para satisfacer las necesidades del usuario, incluyendo así las funcionalidades específicas que se espera que el software tenga, así como cualquier restricción técnica que deba tenerse en cuenta.

9.1.1. Funcionalidades del software

El software a desarrollar tiene que contar con las siguientes funcionalidades:

- **Generación de optotipos:** El software permitirá generar optotipos de diferentes tamaños, color y transparencia según los parámetros definidos por el usuario.
- **Configuración de parámetros de prueba:** El usuario debería poder configurar los parámetros de prueba, como el decremento de tamaño y velocidad del optotipo, distancia de visualización.
- **Elección del tipo de test a ejecutar:** El usuario podrá escoger el tipo de test el cual se obtendrá la evaluación, así como se expone en la sección 1.2.1
- **Registro y almacenamiento de resultados:** El software registrará y almacenará local y automáticamente los resultados de cada prueba realizada por el usuario, permitiendo su posterior consulta y análisis. Estos resultados se darán en valor LogMAR[15], decimal y la velocidad en grados por segundo
- **Interfaz de usuario amigable:** El software contará con una interfaz de usuario fácil de usar, con instrucciones claras y precisas para realizar la prueba. Se añade también un manual de usuario.
- **Exportación de resultados:** El software permite exportar los resultados de las pruebas en formato CSV.

9.2. Arquitectura del software

En esta sección se explica la arquitectura del software, el diseño y las decisiones tomadas en el proceso.

Así como se ha expuesto anteriormente, el desarrollo de este proyecto se ha utilizado *Godot Engine* como herramienta de desarrollo de *software*. Éste cuenta con un editor visual y un lenguaje de programación propio: *GDScript*, con una sintaxis muy similar a *Python*.

La arquitectura del software se ha dividido en tres componentes principales:

- **Lógica de la aplicación.**
- **Interfaz del usuario.**
- **Gestión de datos.**

9.2.1. Lógica de la aplicación

La lógica de la aplicación ha sido implementada en *Godot* utilizando su **sistema de nodos y escenas**. En *Godot*, los **nodos representan las entidades en la escena** y se pueden conectar entre sí para formar una jerarquía de nodos. Cada nodo puede tener un conjunto de variables, señales y métodos. La comunicación entre nodos se realiza mediante señales, que son eventos que se emiten desde un nodo y que pueden ser capturados por otros nodos.

Para la implementación de la lógica de la aplicación se han utilizado principalmente nodos de tipo *Control* y de tipo *Node2D*. Los nodos de tipo *Control* se han utilizado para implementar elementos de la interfaz de usuario, como botones, etiquetas y campos de texto. Los nodos de tipo *Node2D* se ha utilizado para representar el elemento del optotipo.

Para el control del flujo de la aplicación se ha utilizado el sistema de escenas de *Godot*. Las escenas representan conjuntos de nodos y se pueden instanciar y cargar de forma dinámica durante la ejecución de la aplicación.

A continuación se presenta un diagrama con el **flujo de trabajo** de la aplicación de cómo el usuario debería interactuar con el *software*:

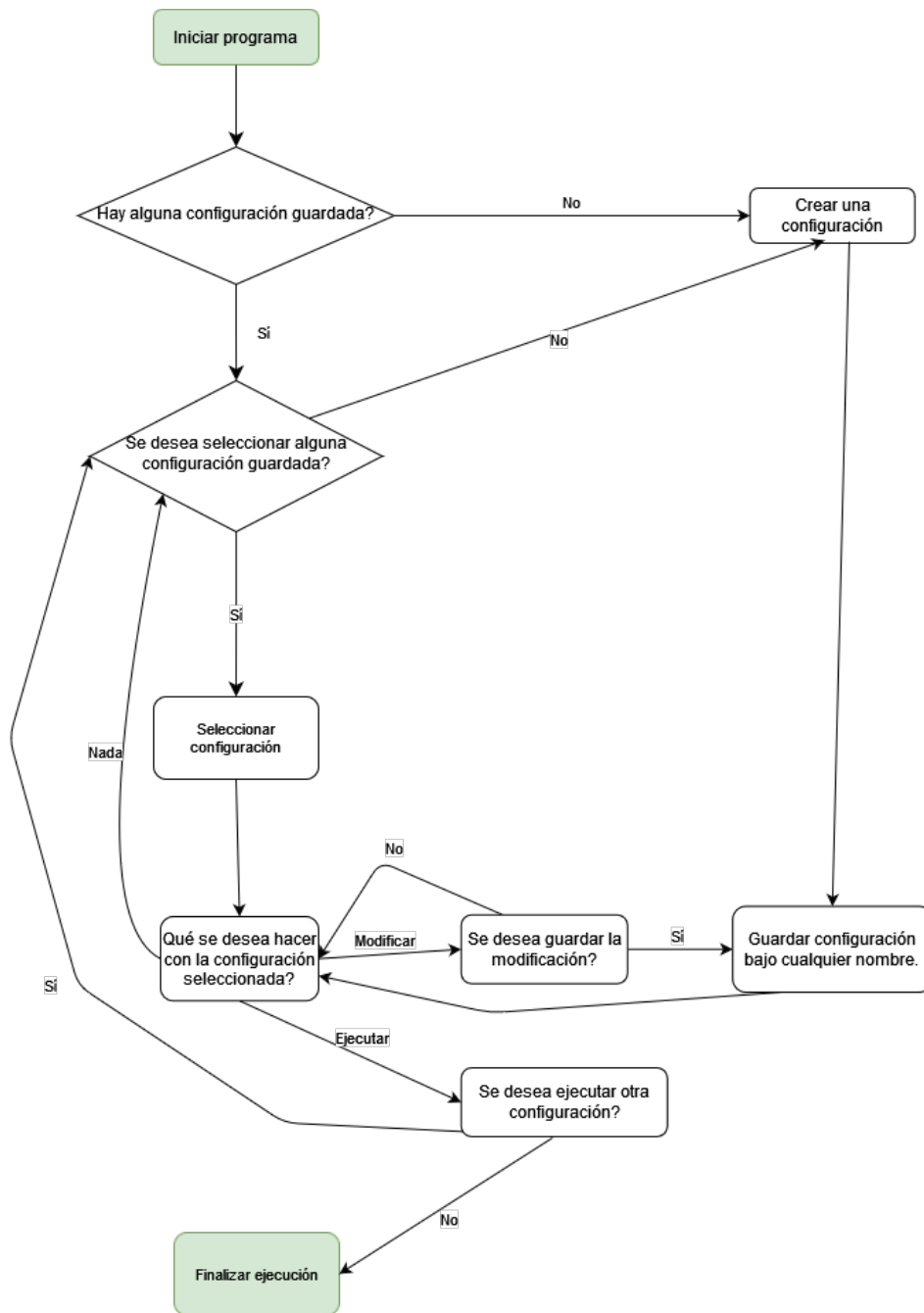


Figura 4: Diagrama realizado con la herramienta draw.io. [16]

Cabe añadir que es importante asegurarse de que los datos introducidos por el usuario sean válidos. Aquí se exponen algunas medidas para asegurarse de esto:

- De forma general, la parte más importante es asegurarse de que los valores introducidos sean números flotantes y no otro valor diferente (una palabra por ejemplo). Por suerte, esta validación se hace de forma nativa en la gran mayoría de los elementos de la interfaz de usuario relacionados a utilizar números, ya que la aplicación no permite al usuario añadir valores no numéricos. Pero haciendo uso de esta dinámica, dicho elemento queda limitado a las especificaciones de éste. Un ejemplo es que el proyecto empezó utilizando un elemento de tipo *SpinBox*, que permite al usuario añadir un número flotante y aumentar/disminuir el valor con unas flechas. Lo malo es que por diseño el número se redondea al entero más cercano y dado que es necesario que se pueda ser lo más específico posible se ha tenido que buscar otra solución. La solución ha sido añadir un elemento de tipo *Textbox* que permite al usuario añadir cualquier tipo de valor y luego vía código se hace la revisión de que el valor introducido sea un número flotante, si éste es flotante, se actualiza el valor, y si éste no es un valor válido, se revierte el valor al último número flotante conocido.
- Otra validación se hace en el momento de guardar una configuración se debe especificar un nombre para su identificación posterior. Este nombre no debe ser una palabra vacía ni empezar por un espacio vacío. Se permite que se utilice un nombre ya utilizado por otro test, con su consiguiente pantalla de confirmación para el usuario preguntando si desea sobrescribir la anterior configuración con dicho nombre. Se muestra a continuación una captura con un mensaje que ayuda al usuario a identificar por qué no puede guardar su configuración con un nombre inválido:

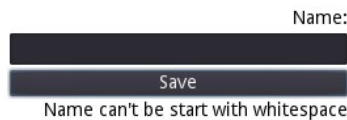


Figura 5: *Captura de pantalla de guardado.*

Durante la ejecución del programa, hay algunas excepciones que se han tenido que controlar para evitar el mal uso de la herramienta.

- ¿Qué pasa si el usuario no ha seleccionado ninguna configuración y pulsa el botón de *Start*? Se ha limitado que dicho botón sea utilizable si se ha seleccionado alguna configuración
- Una configuración es un conjunto de elementos que definen los tests a ejecutar. Cada test define qué valor inicial tiene y en qué momento se ejecutará. En la interfaz gráfica. En la aplicación cada test tiene unos elementos de control, uno de tipo *Label* que denota un *id* mostrando el orden de ejecución y un *SpinBox* que denota el valor inicial de éste. Para añadir un test, es tan sencillo como hacer click izquierdo en el id de alguna de las configuraciones, y éste añadirá un test a la derecha de la configuración de la cuál se ha hecho click. Y para eliminar se hace click derecho en la configuración a eliminar. ¿Pero qué pasa si solo queda un test y se desea eliminar? Pues esto no sería posible, porque al no haber tests disponibles, no se podría hacer click izquierdo de ninguna forma, inhabilitando así la opción de añadir más tests. Esta excepción se ha limitado en la aplicación, prohibiendo al usuario eliminar un test si éste el único que queda.

A continuación se muestra una captura de pantalla con una cadena de tests donde se indica el id y el valor inicial de cada uno de éstos.

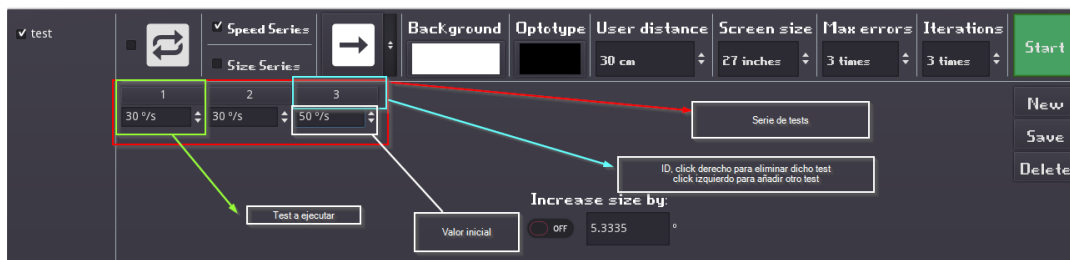


Figura 6: Captura de pantalla, mostrando las partes de un test.

9.2.2. Interfaz del usuario

Desde un principio se diseñó la aplicación como si fuera un videojuego, así que se ha utilizado una tipografía de la temática *8-bit*. La interfaz es grande, para facilitar el uso de ésta, se ha mostrado en verde el botón para empezar la ejecución de la configuración seleccionada y en rojo el botón para finalizar la ejecución del programa. Los valores globales (es decir, los que no mutan en ejecución de cada test) se han agrupado en la parte superior y los valores específicos (los que cambian en función del tipo de test a ejecutar) se han agrupado en la parte central. Si hay muchos valores específicos, el usuario es capaz de hacer scroll horizontal para poder añadir de forma casi infinita los tests que él desee). En la parte izquierda se muestran los nombres de cada configuración y haciendo click, se seleccionará y se podrá ver de forma dinámica los tests configurados, éste tiene un scroll vertical.

A continuación se muestran las diferentes pantallas del usuario y se explican cada uno de los elementos.



Figura 7: Captura de la pantalla de inicio.

En esta captura de pantalla se muestra y se identifican con un número los diferentes tipos de elementos. Se explican a continuación qué función cumple cada uno de dichos elementos:

- **1.** Es un elemento de tipo *Checkbox*, que muestra gráficamente si la opción seleccionada ha sido aplicada o no. En términos de qué funcionalidad cumple bajo el programa, define el comportamiento del optotipo cuando llega al final de la pantalla. Si está activado, el optotipo hará el camino recorrido de forma inversa, es decir se modifica el signo de la velocidad. Si está desactivado, el optotipo hará el camino realizado otra vez.
- **2.** Este grupo de elementos son dos *Checkbox*, técnicamente se podría haber definido solo un *Checkbox* y no dos, pero para facilitar al usuario se ha implementado como dos. Define el tipo de serie a ejecutar, si la *speed serie* o la *size serie*. El comportamiento de estos dos *Checkbox* se podría decir que se comporta como una puerta lógica *XOR*, en ningún momento estos dos pueden estar activados y desactivados a la vez. Vía código se ha dejado claro dicho comportamiento.
- **3.** Este elemento es un *OptionButton* y básicamente es un desplegable con un conjunto de elementos. Estos muestra hacia que dirección se moverá el optotipo. Para facilitar su uso, se ha añadido una imagen en vez de su nombre en el sistema de puntos cardinal.
- **4.** Este ítem es nativo de *Godot* y es un *ColorPickerButton*, permite al usuario definir de forma gráfica un color. Este va ligado con el fondo de pantalla el cuál se ejecutará el test.
- **5.** Este elemento, es similar al elemento **4**, pero a parte se ha añadido el posibilidad de añadir transparencia. Define el color y opacidad del optotipo.
- **6.** Elemento de tipo *SpinBox* que permite al usuario definir a qué distancia de la pantalla se ejecutarán los tests. Este valor es importante definirlo correctamente porque el tamaño y velocidad del optotipo se verán afectados.
- **7.** Elemento de tipo *SpinBox*, necesario para definir el tamaño de la pantalla. Éste no se ha podido detectar de formá automática (la resolución por ejemplo sí), así que es necesario que se ponga de forma manual.
- **8 y 9.** Elementos de tipo *SpinBox*, permiten al usuario definir cuántos errores son permitidos por cada test, si los errores cometidos igualan o superan a dicho límite, no se mostrará ninguna puntuación en dicho test. También permiten definir cuántas iteraciones tendrá cada test.
- **10.** Botón que permite empezar a ejecutar la configuración seleccionada con los diferentes tests configurados.
- **11.** Botón que permite crear una nueva configuración. Si se pulsa mientras hay una configuración modificada no guardada, los datos se perderán.

- **12.** Botón que permite guardar una configuración. No funciona si no hay ninguna configuración seleccionada.
- **13.** Botón que permite eliminar una configuración seleccionada. No funciona si no hay ninguna configuración seleccionada.
- **14.** Inicialmente un *SpinBox*, pero modificado a un *TextBox*. Permite al usuario definir el incremento/decremento de velocidad/tamaño del optotipo para todos los tests bajo una configuración.
- **15** Elemento de tipo *CheckButton*, permite al usuario definir si se quiere especificar que el incremento/decremento sea de tipo porcentual o definir el valor de la unidad a aumentar/disminuir. Para facilitar al usuario su correcta identificación, se añade un *TextLabel* que indica la unidad a aumentar/disminuir, alternando entre 3 valores (^o, ^o/s y %).
- **16.** Elemento de tipo *SpinBox*, permite al usuario definir qué velocidad/tamaño inicial tendrá el optotipo en dicho test. El sufijo cambia según el valor del grupo seleccionado con el número **2**.
- **17.** Botón que permite al usuario identificar el orden de ejecución del test y añadir/eliminar tests al gusto del usuario.
- **18.** Listado de configuraciones posibles, es una lista de *CheckBox*. Hacer click en cualquiera de ellas permite al usuario escoger qué configuración desea seleccionar, para su posterior modificación, eliminación o ejecución.

A continuación se muestra la pantalla de guardado y sus diferentes formas en función del input del usuario.

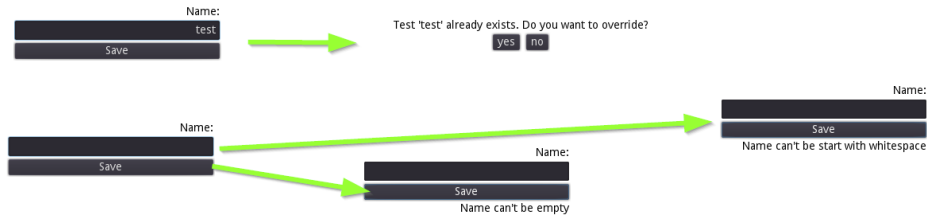


Figura 8: Capturas de las pantallas de guardado de la configuración.

Dicha captura ejemplifica los diferentes comportamientos de la pantalla de guardado.

9.2.3. Gestión de datos

En esta aplicación hay dos tipos de datos:

- Los datos relacionados con la configuración de tests que se desee guardar. Útil para el examinador, así no tendrá que poner cada configuración cada vez que quiera examinar a alguien.
- Datos relacionados con los resultados del test. Estos se muestran en pantalla al finalizar la ejecución y brinda al usuario la posibilidad de guardar los resultados en formato csv, para su posterior análisis y/o incorporarlos a su perfil médico.

Se muestra a continuación la pantalla de guardado de resultados:

```
RESULTS:
TEST:1 LogMar = 1235243, Decimal = 0.058178, Speed = 30°/s
TEST:2 LogMar = 1059152, Decimal = 0.087267, Speed = 15°/s
TEST:3 LogMar = 1059152, Decimal = 0.087267, Speed = 40.000002°/s
TEST:4 LogMar = 1235243, Decimal = 0.058178, Speed = 45.000001°/s
```

Save as CSV Return

Figura 9: Captura de la pantalla de guardado de los resultados.

9.2.3.1 Protección de datos

Para cumplir con las políticas GDPR al guardar un archivo CSV con valores numéricos de una prueba, se deben tomar las siguientes medidas, obtenidas del organismo europeo oficial de GDPR[17]:

- Anonimización de datos: Antes de guardar el archivo, es necesario asegurarse de que los datos personales de los participantes estén anonimizados. Esto significa que los nombres, direcciones, números de identificación y otra información personal identificable se deben eliminar o codificar para que no puedan ser identificados. En este caso, la aplicación no almacena ningún tipo de identificador del usuario.
- Almacenamiento seguro: El archivo csv debe almacenarse en un lugar seguro, preferiblemente en un servidor que esté protegido por medidas de seguridad adecuadas, como *firewalls*, cifrado de datos y autenticación de dos factores. Este programa está desarrollado para que los datos que se deseen guardar, se haga de forma local y deja al usuario la responsabilidad y seguridad de su dispositivo.
- Eliminación de datos: Cuando los datos ya no sean necesarios, se deben eliminar de forma segura. Esto puede incluir la eliminación de los archivos CSV y cualquier copia de seguridad o archivo en caché que se haya creado. La aplicación no se hace cargo de los ficheros manualmente generados ya que no dispone de acceso.
- Consentimiento informado: Es importante obtener el consentimiento informado de los participantes antes de recopilar y procesar sus datos personales. Esto significa que deben ser informados claramente de cómo se utilizarán sus datos y dar su consentimiento explícito para que se utilicen con ese fin. No hace falta ningún tipo de consentimiento del usuario ya que la aplicación no recopila ni procesa ningún tipo de dato personal.

9.3. Implementación del software

En este apartado se describen cómo se han implementado las diferentes características más importantes de la aplicación.

9.3.1. Animación del optotipo

Es necesario hacer una pequeña introducción antes sobre la clase necesarias para la animación del optotipo, más concretamente la clase "Node2D" de Godot. La clase Node2D es una de las clases más básicas y fundamentales en el motor de juego Godot. Es una subclase de la clase Node, que es la base de todas las entidades en Godot[18]. Como su nombre indica, la clase Node2D es utilizada para representar objetos 2D en el juego. Los nodos de esta clase tienen las físicas ya implementadas, lo que significa que pueden ser afectados por la gravedad y las colisiones con otros objetos en el mundo del juego. La clase Node2D proporciona una amplia variedad de métodos para trabajar con objetos 2D, como establecer la posición, escala y rotación, y controlar la transparencia

y visibilidad del objeto. También ofrece un conjunto de propiedades físicas como la fricción, la elasticidad y la masa, que se pueden ajustar para simular el movimiento y la interacción de los objetos en el mundo del juego.

Para la creación de la animación se siguieron los siguientes pasos:

- Se diseñó el optotipo en *Gimp* y se exportó en formato PNG.
- Se importó la imagen a *Godot* y se creó una clase extendiendo *Node2D* para manejar la animación.
- Se crearon variables para almacenar la velocidad, dirección, posición y rotación del optotipo.
- Se definieron fórmulas para ajustar estos parámetros y transformar las unidades de píxeles a unidades de medida relacionadas con la optometría.
- Se utilizó una función que se ejecuta en cada frame del juego para actualizar la posición y rotación del optotipo en función de los parámetros configurados.
- Se implementó la interacción con el usuario mediante temporizadores que activan la animación en momentos determinados y mediante input de teclado que permite avanzar en la ejecución del test.

Para optimizar la animación se puso especial atención en la función que se ejecuta en cada frame del juego. Se buscó que esta fuera lo más sencilla posible para no consumir muchos recursos del ordenador y mantener los fps constantes.

Otro desafío fue solucionar un *glitch* que ocurría si se ejecutaba el *size series* y la velocidad cambiaba de signo en algún momento. Se encontró que esto se debía a un problema con el cálculo de la velocidad en el momento de invertir la dirección. Se resolvió haciendo que el optotipo se parara en ese momento y volviera a empezar con la nueva dirección.

9.3.2. Aplicación de la configuración

La pantalla principal tiene diferentes parámetros que brinda al examinador la oportunidad de crear y guardar diferentes configuraciones a medida. Dicha pantalla principal, en realidad es una clase de *Godot* con diversos elementos de control (botones, cuadros de texto...). Esta clase (y el resto de clases que engloban la aplicación), son hijas de otra clase principal que llamaremos clase "main". Dicha clase "main", permite gestionar el momento en que cada clase hija suya inicia y con qué valores.

Algunas clases son dependientes de otras, así que para comunicarse entre ellas, se hace uso de *signals*: Los signals son un mecanismo en *Godot* que permite la comunicación entre nodos (objetos) de manera eficiente y modular. Cada nodo puede emitir señales (signals) cuando se produce un evento, y otros nodos pueden conectarse a esas señales para responder a ese evento. Los signals son una parte fundamental de la programación

en Godot, ya que permiten una comunicación entre los nodos sin necesidad de acoplarlos directamente.

Por ejemplo, cuando se pulsa el botón de "Start", la pantalla principal envía una señal a señal a la clase *main*, que indica a la clase del Optotipo que es su momento de iniciar. La clase *main*, se encarga de coger los valores configurados en la clase principal y configurar los parámetros necesarios para crear la clase del optotipo. La creación de señales se puede hacer de la siguiente forma:

```
# Definimos la señal que emite el nodo "Player"
signal player_moved(position)
func _ready():
    # Conectamos la señal al método a llamar
    connect("player_moved", self, "on_player_moved")

func move_player(new_pos):
    # Actualizamos la posición del jugador
    position = new_pos
    # Emitimos la señal "player_moved" con la nueva posición
    emit_signal("player_moved", position)

func on_player_moved(new_pos):
    # Este método se llamará cada vez que se mueva el jugador
    print("El jugador se ha movido a la posición:", new_pos)
```

En este ejemplo, la señal *player_moved* se emite cada vez que el jugador se mueve, y se conecta al método *on_player_moved* del mismo nodo. Cuando el jugador se mueve, se actualiza su posición y se emite la señal *player_moved* con la nueva posición como argumento. El método *on_player_moved* se llama automáticamente cada vez que se emite la señal, y se encarga de imprimir la nueva posición del jugador por consola.

Este es solo un ejemplo básico de cómo usar los signals en Godot. Los signals pueden ser mucho más complejos y flexibles, permitiendo una gran variedad de interacciones entre nodos en una escena.

Otro ejemplo más complejo fue, dado el siguiente escenario:

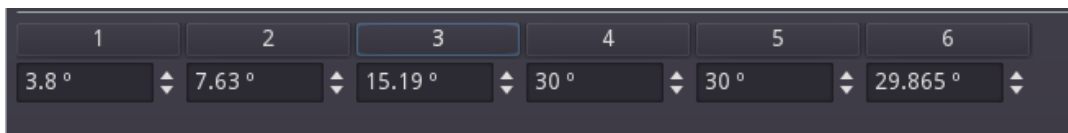


Figura 10: Configuración del número de tests y valores iniciales.

Se puede observar una cadena de elementos en esta captura, donde cada elemento se puede dividir en otros dos subelementos: un botón con un número como identificador, y un *spinBox*. Cuando el usuario hace click puede pasar lo siguiente:

- **Click derecho.** Esto hace que el elemento el cuál se ha hecho click se elimine. Esto es posible a que el elemento seleccionado envía una señal a la clase que gestiona la pantalla de inicio, haciendo que se ejecute una función. Dicha función itera sobre dicha cadena de botones y elimina el botón que tiene la propiedad de *is_Pressed*, la cuál es se implementa en la clase la cuál pertenece el botón. Dicha función también se encarga de volver a enumerar de forma correcta el resto de los botones de cada elemento de la cadena. Aquí se presenta un *snippnet* del código:

```
func _on_VBoxContainer_delete():
    if $Cadena.get_children().size() == 1:
        return
    for button in $Cadena.get_children():
        if button.is_Pressed()==1:
            $Cadena.remove_child(button)
            renumerate_values($Cadena.get_children())
    return
```

Es necesario, que cuando se exija eliminar un elemento, el sistema no deje al usuario hacerlo, ya que si eso pasase, al no haber elementos, no dejaría crear nuevos elementos.

- **Click izquierdo.** Permite el usuario crear un nuevo elemento a la derecha del seleccionado. Esto envía una señal a la clase que gestiona la pantalla de inicio que hace ejecutar la función de añadir un nuevo elemento en el orden correcto y volver a numerar los elementos. Dicho elemento creado, tiene valores por defecto, como el valor del *spinBox* o la señales necesarias para que el nuevo elemento se comporte de forma correcta.

9.3.2.1 Cálculo de resultados

La clase relacionada con la ejecución del test, es la clase Optotipo la cuál se le indican unos valores iniciales y ésta empieza a ejecutarse. En el transcurso de los test, en cada momento se sabe la velocidad y tamaño del optotipo, que son los valores necesarios para dar los resultados. En el momento que el usuario haga un acierto, estos valores se guardan, y si el usuario falla, los resultados no se guardan, pero se deja claro que éste ha cometido un error en dicho test, y si el error es igual o superior al límite establecido, no se dará ninguna puntuación para dicho test y seguirá con el siguiente test. Cuando se acaba el test, se hace una media con los valores obtenidos y se calcula la agudeza visual en valor tipo logMAR y decimal, así como la velocidad. Los valores obtenidos se guardan hasta que finaliza todos los tests. Cuando el usuario realiza todos los tests configurados, se muestra por pantalla los resultados obtenidos y brinda la oportunidad de guardarlos en formato *csv*. El cálculo es el siguiente:

$$\log MAR = \log_{10} s * 60 \quad (1)$$

$$decimal = 10^{-\log MAR} \quad (2)$$

Donde s es el valor del tamaño del optotipo en grados visuales. La unidad de grado visual se puede calcular con los parámetros proporcionados por el usuario, que va en función de cuán lejos está el usuario de la pantalla y el tamaño de la diagonal de la pantalla en pulgadas.

9.4. Fase de pruebas y errores

A medida que se ha ido desarrollando la aplicación se ha ido testeando con la ayuda del codirector Marc Argilés y la Dra. Lluïsa Quevedo las funcionalidades que han ido creciendo durante el transcurso de este trabajo, gracias a sus comentarios y correcciones se ha podido verificar la correcta utilización del programa. A nivel de desarrollador, lo más complicado ha sido quizás verificar que los datos de la velocidad y tamaño fueran los correctos, esto ha hecho que se calculen unos valores base para la velocidad y tamaño, y a tiempo real medirlos durante la ejecución del programa para ver si eran correctos o no. Con un temporizador y literalmente una regla, se han podido verificar los valores del tamaño y velocidad.

Por último y no por eso menos importante, se han utilizado *breakpoints*, marcadores de línea de código que permiten parar la ejecución del programa y así poder ver qué variables y qué valores hay en ese momento. Esto era posible a la interfaz de programación de *Godot*.

9.5. Documentación para el usuario

Para el correcto uso de la aplicación se facilita un manual de usuario para ejecutar el test. Se expone en el anexo

10. Conclusión

En resumen, el desarrollo del software ha sido un éxito total. Gracias a la buena planificación y gestión del proyecto, se han resuelto todos los problemas previstos y se han obtenido resultados favorables que superaron las expectativas iniciales. La aplicación ha sido rigurosamente probada y validada para asegurar su correcto funcionamiento, lo que nos permite confirmar que el grado de cumplimiento del proyecto ha sido del 100 %. El software desarrollado es único en su género, ya que no existe otra aplicación que realice esta función específica. Es importante destacar que el impacto de esta aplicación será muy positivo, ya que será de gran ayuda tanto para deportistas como para personas mayores, y se utilizará en otras universidades de oftalmología. Además, el software es *open source*, lo que permite a cualquiera modificarlo a su gusto.

La interfaz de usuario ha sido uno de los aspectos fundamentales en el desarrollo de la aplicación, logrando crear una experiencia de usuario satisfactoria para todo tipo de usuarios. La interfaz es accesible y fácil de usar, gracias a su diseño intuitivo y a la inclusión de una guía de usuario detallada. Además, se han implementado diversas funcionalidades que optimizan la navegación y uso de la aplicación, lo que garantiza un alto grado de satisfacción por parte del usuario. Es importante destacar que se ha trabajado para cumplir con las especificaciones y requerimientos previamente establecidos en el diseño de la interfaz, alcanzando un grado de cumplimiento del 100 %, lo que asegura la efectividad y usabilidad de la aplicación.

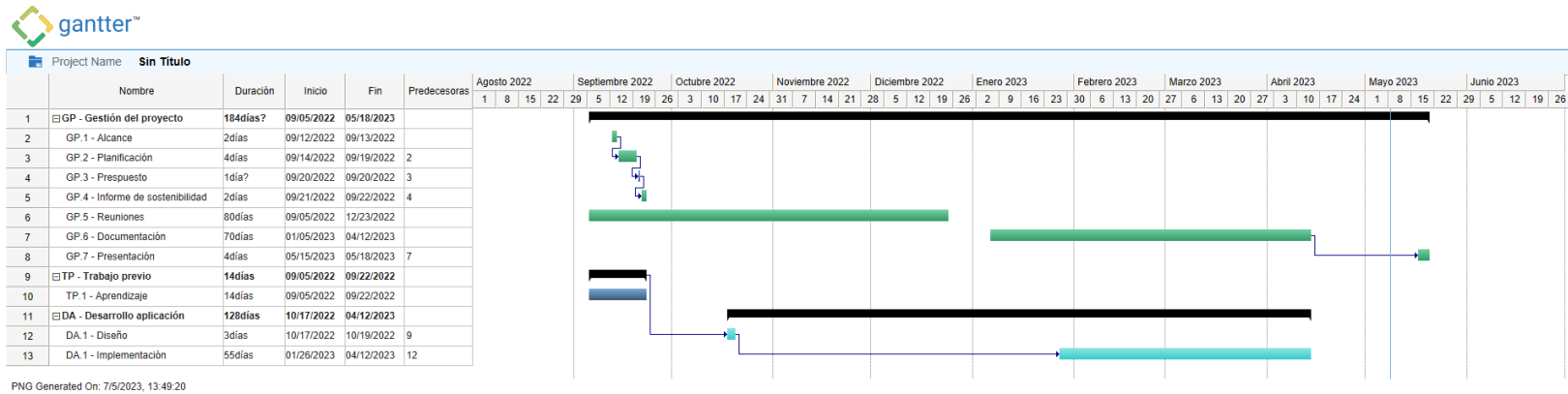
En conclusión, me siento muy satisfecho con los resultados obtenidos en este proyecto de desarrollo de software. Se han cumplido todas las especificaciones y requisitos previamente establecidos, y se ha creado una aplicación única y de gran ayuda para la sociedad. Además, el software es *open source*, lo que permite a cualquiera modificarlo a su gusto y contribuir al proyecto. Se ofrece la oportunidad de contactar con el autor para cualquier sugerencia o mejora que se quiera añadir a la aplicación.

11. Bibliografía

- [1] Federico. *¿Qué es la optometría deportiva?* Mar. de 2018. URL: <https://www.federopticos.com/blog-optometria-deportiva/>.
- [2] Elisa Aribau. *El test de Snellen y la agudeza visual*. Mar. de 2018. URL: <https://www.elisaribau.com/test-snellen-la-agudeza-visual/>.
- [3] *Dictionary of visual science ; edited by David Cline, Henry W. Hofstetter, John R. Griffin*. 1980.
- [4] Lluïsa Quevedo et al. «A novel computer software for the evaluation of dynamic visual acuity». En: *Journal of Optometry* 5.3 (2012), págs. 131-138. ISSN: 1888-4296. DOI: <https://doi.org/10.1016/j.optom.2012.05.003>. URL: <https://www.sciencedirect.com/science/article/pii/S188842961200057X>.
- [5] Administrator. *Optotipo Anillo disco palomar 'Optotipo Universal'*. URL: https://www.centrospalomar.com/index.php?option=com_content&view=article&id=246%3Aoptotipo-anillo-disco-palomar-qoptotipo-universalq-&Itemid=&lang=galego.
- [6] *Trello ayuda a Los equipos a sacar el trabajo adelante*. URL: <https://trello.com/es>.
- [7] *Where the world builds software*. URL: <https://github.com/>.
- [8] *1 cloud-based project management software*. URL: <https://www.gantter.com/>.
- [9] Unity Technologies. URL: <https://unity.com/es>.
- [10] Godot Engine. *Free and open source 2D and 3D game engine*. URL: <https://godotengine.org/>.
- [11] *Requirements test*. URL: <https://www.systemrequirementslab.com/cyri/requirements/godot-engine/19806>.
- [12] *Overleaf, online latex editor*. URL: <https://www.overleaf.com/>.
- [13] Sergio Luján Mora. *Herramientas para la Investigación*. URL: <http://desarrolloweb.dlsi.ua.es/cursos/2015/herramientas-investigacion/que-es-latex>.
- [14] *ESGuiaHaysdelMercadoLaboral*. URL: <https://www.hays.es/documents/63345/29167077/ESGuiaHaysdelMercadoLaboral2022.pdf>.
- [15] Jaime García Aguado et al. *Valoración de la Agudeza Visual*. URL: https://scielo.isciii.es/scielo.php?script=sci_arttext&pid=S1139-76322016000300019.
- [16] *Security-first diagramming for teams*. URL: <https://www.diagrams.net/>.
- [17] URL: <https://gdpr.eu/>.
- [18] URL: https://docs.godotengine.org/en/stable/classes/class_node2d.html.

12. Anexos

Anexo A. Diagrama de Gantt



Anexo A: Diagrama de Gantt, producción propia.

Anexo B. Manual del usuario

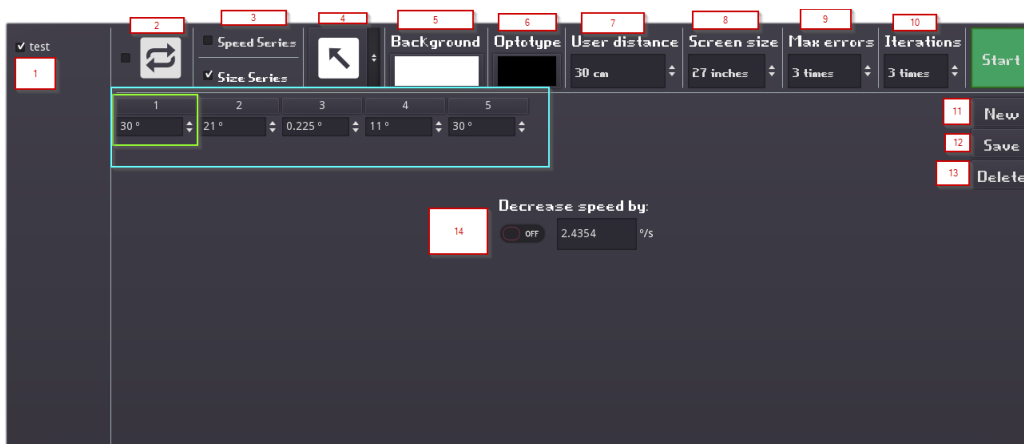
Este documento está destinado para un especialista en el campo de la oftalmología. Algunos términos de rasgo científico no son explicados aquí.

Introducción

Esta aplicación permite el cálculo de la agudeza visual dinámica a través de una serie de movimientos realizados por un optotipo, estos tests pueden ser de dos tipos:

- **size series**, donde el tamaño es fijo en todas las iteraciones del test, y se añade un decremento de velocidad en porcentaje o en grados por segundo.
- **speed series**, donde la velocidad es fija en cada momento y lo único que varía es el tamaño del optotipo, que cada vez irá siendo más grande. El aumento se define en porcentaje o en grados.

Hace falta definir que una configuración se compone por uno o más tests, donde cada test comparte algunos valores, como el número de veces que se tiene que ejecutar el test antes de pasar al siguiente test, o el número de errores que se pueden cometer por test. En la siguiente captura se muestra un ejemplo de una configuración y se explican algunos de sus atributos, así como su funcionamiento:



Pantalla de inicio.

Se ha seleccionado la configuración con nombre test (1), el optotipo cuando salga de la pantalla volverá a su posición inicial (2), los tests son de tipo *size series* (3), el optotipo se desplazará de sureste a noroeste (4), el color del fondo será blanco (5) y el color del optotipo negro (6), la distancia del usuario es de 30 cm (7), el tamaño de la pantalla es de 27 pulgadas (8). Un test se puede ver respresentado en el rectángulo verde, donde el número superior indica el orden del test (es decir en qué momento se ejecutará), y el número inferior representa el tamaño de dicho test. En este caso el test marcado en la zona verde, se ejecutará 3 veces (10) con un máximo de 3 errores (9) con

un tamaño de 30° y una velocidad de $2.4354^\circ/\text{s}$ (14).

Se puede crear una configuración haciendo click en el botón "new" (11), se puede guardar con el botón "save" (13), y se puede eliminar con el botón "delete" (13).

Se puede añadir un test haciendo click izquierdo en el número que indica el orden del test (número superior del rectángulo verde), a su vez, también se puede eliminar un test haciendo click derecho en el número que indica el orden del test (número superior del rectángulo verde).

¿Qué debe hacer el usuario?

El usuario deberá escoger con el teclado numérico (siendo el número 8 el norte, el 9 noreste, el 6 el este y así consecutivamente) la orientación del agujero en el optotipo.

Resultados

Los resultados se obtienen al finalizar los tests. Se muestran por pantalla la media de cada test en función de la agudeza visual, en logMAR y en decimal, y de la velocidad, en grados por segundo.

El sistema permite exportar los resultados en formato csv.