UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Implementación de un simulador de computador cuántico

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Carlos Gálvez Aucejo

Tutor: Carlos Herrero Cucó

Curso 2020-2021

Resum

La computació quàntica és un àrea d'investigació que estudia els algorismes i sistemes que usen les propietats de la física cuàntica per a solucionar problemes matemàtics complexos o costosos. El principal avantatge dels computadors quàntics és la capacitat dels qubits de mantindre una superposición de tots dos estats {0,1}. Gràcies a això, s'aconsegueix avaluar i guardar totes les possibles solucions simultàniament que dona com a consequència una millora exponencial en l'obtenció dels resultats. El treball consisteix a desenvolupar un simulador de computació quàntica que permet executar circuits quàntics amb diverses portes lògiques i diversos qubits en un ordinador clàssic. L'aplicació del simulador és d'escriptori, està acompanyada d'una interfície gràfica, desenvolupada amb el llenguatge Python i té un caràcter educatiu, il·lustrant els fenòmens de la quàntica. Pots simular els teus propis circuits arrossegant les portes lògiques, guardar-los en l'ordinador o llançar-los en maquinari quàntic real de proveïdors com IBM. Es comparen les característiques dels diferents sistemes quàntics actuals d'importants empreses tecnològiques.

Paraules clau: simulador quàntic, computador quàntic, cúbit, Python, Qiskit, IBM

Resumen

La computación cuántica es un área de investigación que estudia los algoritmos y sistemas que usan las propiedades de la física cuántica para solucionar problemas matemáticos complejos o costosos. La principal ventaja de los computadores cuánticos es la capacidad de los cúbits de mantener una superposición de ambos estados {0,1}. Gracias a esto, se consigue evaluar y guardar todas las posibles soluciones simultáneamente cuya consecuencia es una mejora exponencial al obtener los resultados. El trabajo consiste en desarrollar un simulador de computación cuántica que permite ejecutar circuitos cuánticos con varias puertas lógicas y varios cúbits en un ordenador clásico. La aplicación del simulador es de escritorio, está acompañada de una interfaz gráfica, desarrollada con el lenguaje Python y tiene un carácter educativo, ilustrando los fenómenos de la cuántica. Puedes simular tus propios circuitos arrastrando las puertas lógicas, guardarlos en el ordenador o lanzarlos en hardware cuántico real de proveedores como IBM. Se comparan las características de los distintos sistemas cuánticos actuales de importantes empresas tecnológicas.

Palabras clave: simulador cuántico, computador cuántico, cúbit, Python, Qiskit, IBM

Abstract

Quantum computing is an area of research that studies algorithms and systems that use the properties of quantum physics to solve complex or hard mathematical problems. The main advantage of quantum computers is the ability of qubits to maintain a superposition of both {0,1} states. Thanks to this, it is possible to evaluate and store all possible solutions simultaneously, the consequence of which is an exponential improvement in obtaining the results. The work consists of developing a quantum computing simulator that allows running quantum circuits with several logic gates and several qubits on a classical computer. The desktop application of the simulator is accompanied by a graphical interface, developed with the Python language and has a teaching approach, illustrating the phenomena of quantum. You can simulate your own circuits by dragging the logic gates, save them in the computer or launch them on real quantum hardware

from providers such as IBM. It compares the characteristics of different current quantum systems from major technology companies.

Key words: quantum simulator, quantum computer, qubit, Python, Qiskit, IBM

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Además de la computación clásica, donde los microprocesadores son más pequeños a medida que pasa el tiempo derivando en problemas como el efecto túnel, existe la computación cuántica. La computación cuántica es un campo joven donde se emplean leyes y propiedades del mundo cuántico, teniendo que apartar, en ocasiones, las intuitivas ideas clásicas. A principios de los años 80, Paul Benioff propuso la idea de un computador cuántico e incluso se planteó que este tipo de computación, basada en leyes de la mecánica cuántica, sea capaz de hacer cálculos complejos de manera más rápida que con la computación clásica. El físico David Deutsch presentó en 1985 un marco para la computación cuántica que describía la forma de los algoritmos cuánticos [1]. Junto a Richard Jozsa formalizaron un algoritmo que resolvía el problema planteado en un solo paso utilizando un computador cuántico, mientras que uno clásico supondría un orden de $O(n)$ [2]. Más tarde, Seth Lloyd, físico americano, planteó una manera de llevar a cabo un computador cuántico funcional basándose en un sistema que enviase pulsos que representasen un estado cuántico. En 1994, el matemático Peter Shor expuso un algoritmo para factorizar números grandes en factores primos [7] , algo muy costoso para un computador clásico. Sin embargo, en un ordenador cuántico funcional y con la suficiente potencia de cálculo esto se podría reducir hasta el punto de poner en peligro RSA, algoritmo sobre el que se sustenta la seguridad de la mayoría de las comunicaciones en Internet. Se habla de una mejora exponencial con respecto a un ordenador clásico. David DiVincenzo presentó en el año 2000 un conjunto de premisas que se deben cumplir a la hora de crear un ordenador cuántico. [6]

En comparación con el nacimiento de la computación clásica y sus primeros ordenadores, a los que muy pocos tenían acceso, en la actualidad se dispone de una gran cantidad de simuladores y sistemas cuánticos reales sobre los que ejecutar nuestros circuitos. Muchas empresas importantes como Microsoft, IBM o Google han destinado una gran cantidad de recursos para la investigación de este sector, abriendo plataformas para el público general donde se ofrece documentación de todo tipo: vídeos explicativos, libros de texto, ejercicios planteados, tutoriales de cómo utilizar sus propias herramientas y más. Estas empresas han tomado caminos diferentes y luchan entre sí por ser las más avanzadas que, dependiendo de la implementación física en las que se sustente, tendrán sus propias fortalezas y sus retos a mejorar. Sin embargo, uno de los objetivos del sector es común: intentar demostrar la supremacía cuántica. Se discute si ya ha sido conseguida esta ventaja cuántica por Google [13] mientras meses antes IBM presentaba su primer ordenador cuántico comercial. Y como en todas las ideas revolucionarias, también existen algunos detractores de la escalabilidad de la computación cuántica, como así se autodeclara el matemático Gil Kalai. [15]

1.1 Motivación

La computación cuántica es un tema actual, en auge y que llama mucho la atención al lector. Se pueden leer muchos titulares extremadamente llamativos en distintos medios que tratan de alarmar o impresionar con la potencia computacional que puede ofrecer esta vertiente. Una de las principales razones por las que me adentré en este campo era intentar comprender qué había realmente detrás de esas afirmaciones. Poder estar al tanto de las nuevas noticias y entender los avances o problemas que surgen es algo que apasiona. Además también buscaba profundizar más con el fin de tener la capacidad de resolver las dudas más comunes de compañeros, amigos o familiares sobre el tema.

Por otro lado, intentar dar más visibilidad al campo acercando a gente nueva puede ser muy enriquecedor para todos y es algo que ilusionaría. También a nivel laboral, se valorará positivamente el hecho de haber investigado previamente parte de la computación cuántica y la realización de una aplicación sobre esto.

Finalmente, poder emplear gran parte de los conocimientos adquiridos a lo largo de la carrera en un trabajo final sobre un tema tan llamativo es otro de los motivos.

1.2 Objetivos

El objetivo general de este trabajo de final de grado es crear desde cero una aplicación de escritorio donde se simulen circuitos cuánticos creados por el usuario con el fin de acercar el mundo de la computación cuántica a más gente y todo de manera didáctica. Como objetivos más específicos, se busca:

- Describir las bases de la computación cuántica y sus pecularidades en contraposición con la clásica.
- Analizar los distintos servicios que ofrecen las grandes empresas del sector e identificar sus diferencias.
- Desarrollar una aplicación de escritorio con Python.
- Examinar los resultados de una simulación de un circuito cuántico.
- Lanzar un circuito diseñado con la aplicación en los servicios de otras empresas.
- Emplear una metología ágil como forma de trabajo durante el desarrollo.

Como objetivo transversal y personal se desea plasmar correctamente los conocimientos obtenidos durante toda la carrera.

1.3 Estructura de la memoria

En la estructura de la memoria se mostrará un índice con una breve descripción de los distintos capítulos y secciones del trabajo.

En el primer capítulo se encuentra la introducción al tema del trabajo acompañado de la motivación, donde se verán las razones por las que se ha elegido este TFG, y los objetivos que se buscan lograr con el trabajo.

En el capítulo 2 se da a conocer la situación actual de la computación cuántica a través de una serie de apartados: los distintos tipos de procesadores cuánticos que existen

revisando su implementación, la diferencia entre un simulador y un ordenador cuántico con hardware real y una comparativa descriptiva entre los servicios en la nube más importantes del sector. Además se analizan cuáles son los siguientes pasos en el ámbito de la investigación y se exponen cuales son los argumentos que pretenden demostrar la dificultad de la computación cuántica. Finalmente se ofrece un apartado donde se presentan los conceptos más interesantes de la computación cuántica desde un punto de vista técnico.

En el capítulo 3 se realiza un análisis del problema con varias de las técnicas vistas en las asignaturas del grado. Tras este análisis de requisitos, se busca evaluar la viabilidad del proyecto mediante una matriz DAFO, un modelo de negocio. Finalmente se plantea un Lean Canvas para recoger de manera gráfica gran parte de la información anterior.

En el capítulo 4 se muestra el diseño del sistema a desarrollar. En un primer apartado se abarca la arquitectura desde un nivel más alto y en el diseño detallado se profundiza en lo anterior, indicando las clases del sistema y las relaciones entre ellas. Además se incluye otro subapartado que muestra las tecnologías que se han utilizado para llevar a cabo este proyecto. También se incluyen unas capturas de los prototipos de la interfaz de usuario.

En el quinto capítulo ya se puede revisar el desarrollo de la idea inicial. En primer lugar se definen los flujos de trabajo que se van a utilizar, después se plantea una planificación temporal detallada, basada en MVP, a seguir durante el desarrollo. Finalmente, en el apartado de implementación se detallarán aspectos más técnicos de cómo se han usado las tecnologías sobre el proyecto para llegar a la solución.

En el capítulo 6 se trata de cerrar el trabajo realizado presentando las conclusiones del mismo: si se han logrado los objetivos establecidos y las relaciones que hay entre el TFG y las asignaturas del grado.

El capítulo 7 muestra una lista de mejoras o flecos por cubrir como trabajo a futuro y funcionalidades que se puede implementar en el proyecto.

Adicionalmente, al final del presente documento se ubica la bibliografía y un glosario donde se puede consultar el significado de algun término específico del dominio. También hay un apéndice con una breve descripción del funcionamiento de la aplicación acompañado de capturas.

CAPÍTULO 2
Estado del arte

En este capítulo se recoge información de la situación actual de la computación cuántica y se disponen estos conocimientos en diversos apartados. Aquí se revisan las distintas implementaciones de procesadores cuánticos y se determinan las diferencias entre simulador y computador cuántico. También se realiza una comparativa de los simuladores cuánticos disponibles de las grandes empresas, se exponen los pasos que se buscan dar en el área de investigación y los argumentos en contra de la computación cuántica. Adicionalmente, se presentan conceptos relacionados desde un prisma más técnico.

2.1 Procesadores cuánticos actuales y sus implementaciones

Hay distintos tipos de procesadores cuánticos dependiendo de su implementación física. Las más empleadas son la fotónica y la superconductora, pero también existe la computación cuántica de iones atrapados y basadas en el silicio.

La primera, también conocida como Linear Optical Quantum Computing (LOQC) permite, bajo ciertas condiciones, computación cuántica universal. Utiliza fotones como portadores de la información, instrumentos ópticos para procesar la información y detectores de fotones y memorias cuánticas para detectar y almacenar la información. La ventaja que tiene esta implementación es que la unidad de información cuántica (el fotón) está potencialmente libre de decoherencia cuántica. La contraparte es que los fotones no se relacionan entre sí de manera natural y estas interacciones son necesarias al aplicar puertas de dos cúbits. El ordenador basado en puertas con más cúbits hasta el momento es el Jiuzhang, desarrollado por la Universidad de Ciencia y Tecnología de China, con 76. Fue lanzado en 2020, y en diciembre de ese mismo año se convirtió en el primer computador cuántico que, utilizando a la implementación fotónica, consigue la ventaja cuántica. Sin embargo, se desconoce su porcentaje de fidelidad y su arquitectura no es fácilmente escalable.

La computación cuántica superconductora es otra de las implementaciones. En este caso, está basada en circuitos electrónicos superconductores, es decir que se fabrican con materiales superconductores y funcionan a frecuencia de microonda. En el año 2014 se consiguió desarrollar un sistema cuántico superconductor con una puerta lógica de 2 qubits (usando 5 qubits) con una alta fidelidad de 99.4 % [5]. El mayor logro conseguido recientemente fue en octubre de 2019 cuando se demostró por primera vez, mediante un computador con esta implementación, la ventaja cuántica [13]. Esto se alcanzó con uno de los computadores cuánticos a la cabeza del sector: el Bristlecone del equipo de Google Quantum AI Lab. Dispone de 72 cúbits con un porcentaje de error menor al 1 % en todo tipo de operaciones y se lanzó en marzo de 2018.

En vista del gran rendimiento, muchas empresas tecnológicas como Google, IBM o Microsoft está luchando por posicionarse en la computación cuántica. Los cúbits superconductores tienen las siguientes ventajas: son altamente diseñables, escalables y fáciles de emparejar y controlar (medir y operar sobre ellos). Todo esto hace que la computación cuántica superconductora sea la aplicación mejor posicionada para escalar la computación cuántica.

Es interesante mencionar una noticia reciente a la publicación del trabajo que apunta a un fuerte avance en cuanto a la tecnología de espines de silicio. Y es que científicos de la Universidad de Nueva Gales del Sur, Australia, han conseguido generar un campo electromagnético uniforme capaz de hacer resonar sobre millones de espines de electrones de los cúbits. Todavía falta hacer pruebas con esta hipótesis pero de confirmarse solucionaría uno de los desafíos más importantes de la computación cuántica.

2.1.1. Basado en puertas universales vs Temple cuántico

Hasta ahora se han expuest computadores cuánticos basados en el modelo de circuitos cuánticos y puertas lógicas. Además de las implementaciones ya vistas anteriormente existe otro tipo de clasificación del diseño de los procesadores cuánticos. Las dos opciones que se presentan son ordenadores cuánticos universales basados en puertas y de temple cuántico. Estas alternativas tienen distintos objetivos ya que, a pesar de fundarse en el mismo hardware y concepto, cada una es útil para un tipo de tarea o problema.

Ordenador cuántico basado en puertas universales

Este tipo de computadores tienen mayor cantidad de aplicaciones y se basan en construir cúbits de alta fiabilidad. La idea es emplear operaciones con circuitos o puertas lógicas para construir algoritmos complejos. Además, es necesario que estas puertas sean reversibles, a diferencia de las clásicas para no perder información y no romper la superposición y el entrelazamiento cuántico.

Ordenador de temple cuántico

Los temples cuánticos (*Quantum Annealing*) tienen un mejor rendimiento en problemas donde hay muchas posibles soluciones y se busca una lo suficientemente buena. Un ejemplo de problema adecuado para quantum annealers es el problema del vendedor viajero. D-Wave Advantage es el computador cuántico más avanzado de este tipo con 5760 cúbits. La clave para que sean tan eficientes en ciertos problemas es el *layout*, que se diseña y construye con un formato en el que D-Wave pueda entenderlo eficientemente.

2.2 Diferencias entre simulador cuántico y ordenador cuántico

Un simulador cuántico [9] es un sistema que utiliza los efectos cuánticos con el fin de responder a ciertos problemas físicos. La simulación suele ser el primer paso de un proceso donde el siguiente y último paso sería comparar el modelo físico con un sistema físico real. La simulación es parte de método científico. De otra manera, un simulador cuántico muestra información sobre una función matemática relacionada a un modelo físico. Si se comparan los resultados obtenidos del simulador con un sistema real determinado, la simulación servirá para decidir si el modelo matemático es lo suficientemente preciso para considerar que el simulador se ajusta al sistema cuántico de referencia. Además mientras que en un simulador se puede volver a estados anteriores y visualizarlos,

en un sistema con hardware real en cuanto se realiza una medición del cúbit, este pierde la superposición y se alinea a uno de los dos estados.

Un sistema real, en este caso un ordenador cuántico, tiene muchas similitudes con un simulado, sin embargo, la complejidad aumenta. Por ejemplo, para proteger los cúbits de interferencias hay que aislarlo correctamente y mantenerlo a una temperatura cercana al 0 absoluto (-273,15°C) logrando así mayores tiempos de coherencia cuántica. Las condiciones necesarias para construir un computador cuántico se recogen en los criterios de DiVincenzo [6] y son los siguientes:

1. Un sistema físical escalable con cúbits bien caracterizados.
2. La capacidad de inicializar el estado de los cúbits a un estado fiducial simple, como $|000\rangle$.
3. Tiempos de coherencia relevantemente largos, mucho más largos que el tiempo operacional de las puertas lógicas.
4. Un conjunto de puertas lógicas universales.
5. La capacidad de medida sobre un cúbit en específico.

Además existen dos puntos más que son necesarios para la comunicación cuántica.

i. La capacidad de interconverttir cúbits estacionarios y voladores.

ii. La capacidad de trasmitir fielmente cúbits voladores entre ubicaciones específicas.

2.3 Comparativa de simuladores gratuitos

En [10] se encuentra un listado extenso de los diferentes proyectos open source de Quantum Software, siendo muchos de ellos simuladores de computadores cuánticos.

En este repositorio se pueden encontrar librerías full-stack donde se pueden destacar frameworks de compañías importantes como Qiskit de IBM, Q# de Microsoft, Braket de Amazon o Cirq de Google. A continuación se analizarán los principales competidores.

2.3.1. Qiskit

Es un SDK de código abierto para trabajar en Python con computadores cuánticos al nivel de pulsos, circuitos y módulos de aplicaciones. Acelera el desarrollo de aplicaciones cuánticas ofreciendo un conjunto de herramientas con las que se podrán interactuar con sistemas cuánticos y simuladores. Para desarrollar código cuántico existen muchas opciones y a continuación, se mencionan algunas de estas en forma de paquetes:

- Qiskit [11] permite una investigación y desarrollo fácil para industrias específicas que pueden ser potenciadas gracias a la cuántica. Existen paquetes para optimización de problemas, finanzas, machine learning o química.
- También tiene una colección de algoritmos cuánticos para llevar a cabo investigaciones sobre como resolver ciertos problemas en distintos dominios con circuitos poco complejos. Algunos ejemplos son el algoritmo de Búsqueda de Grover, la factorización de Shor o algunos más didácticos como el algoritmo de Deutsch-Jozsa.

- Otro apartado es el conjunto de herramientas que ofrece para experimentar con el ruido que impacta en sus dispositivos cuánticos, como por ejemplo controlar el error de las puertas lógicas cuánticas. Tiene un paquete destinado al análisis, otro de verificación de puertas y circuitos pequeños y un último.

- Además, gracias a otro conjunto de herramientas se pueden componer programas cuánticos al nivel de circuitos y pulsos, optimizándolos para el procesador cuántico físico concreto y sus especificaciones.

- Finalmente, dado el diseño modular de Qiskit, se pueden desarrollar extensiones.

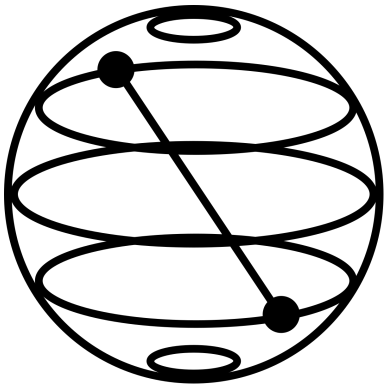


Figura 2.1: Logo de Qiskit.

Para ejecutar el código cuántico, Qiskit ofrece dos opciones:

1. Simulación con un framework de simulador de altas prestaciones. Se ofrecen simuladores optimizados en C++ alojados en la nube de IBM donde se puede ejecutar circuitos y herramientas para construir modelos muy personalizables de ruido para conseguir simulaciones más realistas donde entran en juego los errores de los sistemas reales.

2. Ejecución de circuitos en sistemas cuánticos reales.

En la nube de IBM se nos facilita un compositor de circuitos con interfaz gráfica. Se pueden arrastrar al panel central algunas puertas lógicas preconstruidas. A medida que se insertan y combinan puertas se pueden ver las probabilidades de obtener ciertos valores en los qubits en el panel de abajo a la izquierda. Abajo a la derecha de las probabilidades se ve lo que sería algo similar a la esfera de Bloch, la Q-Sphere donde se ve una vista global del estado cuántico de hasta 5 qubits en la base computaciones. Para acabar, al margen derecho de la pantalla está el código equivalente al circuito creado, incluyendo un acceso directo al Quantum Lab. Ahí se puede escribir código de Qiskit, ecuaciones y visualizaciones en el entorno de desarrollo de una libreta Jupyter.

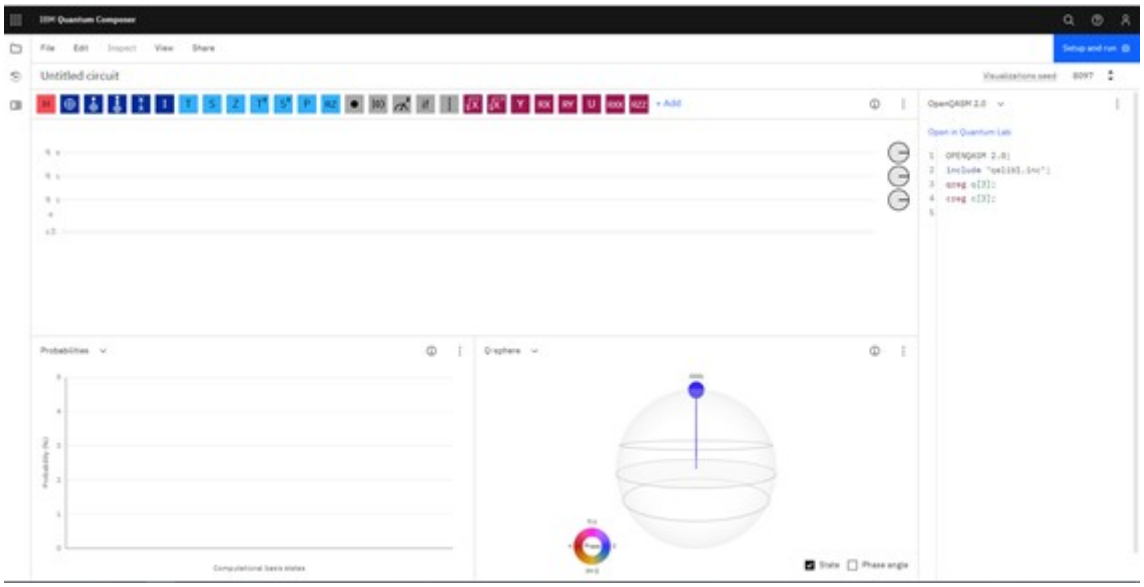


Figura 2.2: Vista general del QuantumComposer de IBM Quantum

Existe un listado de los servicios a los que se pueden mandar los trabajos de circuitos cuánticos una vez completado el registro en la platafoma. Hasta el momento hay disponibles 8 sistemas reales y 5 simuladores de distinto propósito. Sin embargo, siendo miembro con permisos superiores se podría tener acceso al sistema *ibmq_montreal* de 27 qubits y 128 puntos de volumen cuántico. Este sería el más potente hasta la fecha de publicación al que se tiene un acceso universal.

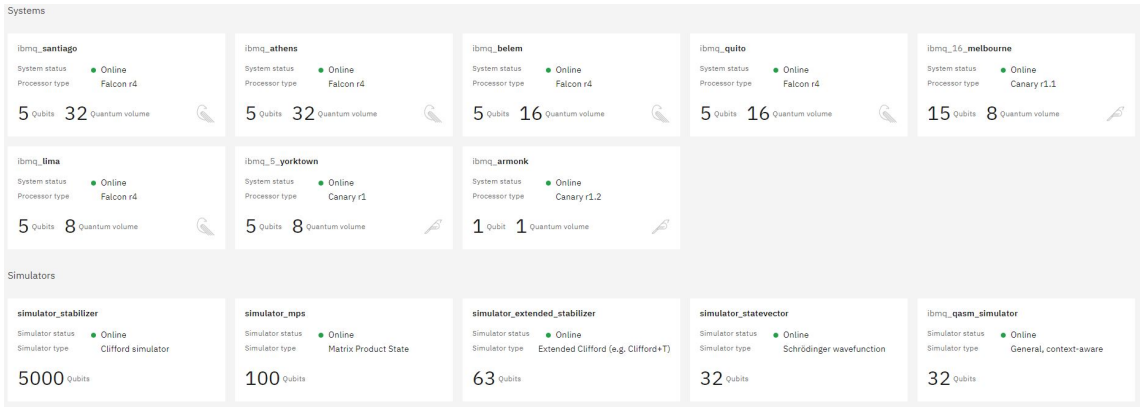


Figura 2.3: Listado de servicios ofrecidos por IBM

Finalmente, hay que destacar que Qiskit tiene una de las comunidades más grandes y activas del sector.

2.3.2. Cirq - Google Quantum AI

Cirq[12] es un framework de código abierto para programación de computadores cuánticos que funciona con Python. Permite escribir, manipular y optimizar circuitos cuánticos para después lanzarlos en simuladores o sistemas reales. Google ofrece una plataforma donde poder acceder a los recursos computacionales cuánticos. De manera similar a IBM, Google permite lanzar programas en simuladores y en procesadores cuánticos como Sycamore, que tiene 51 cúbits y con el que se ha demostrado la ventaja cuántica.[13].

A través de Cirq, se ofrece multitud de opciones donde ejecutar el código. Empezando por el hardware de Google, actualmente no es tan sencillo lanzar circuitos en sus procesadores cuánticos ya que está restringido a personas con invitación o rellenando un cuestionario. Con este cuestionario [**?**] tendrás acceso a usar su API Quantum Engine que ofrece distintos backends, computadores cuánticos o simuladores. También están disponibles, en cierta medida, hardware de otras empresas donde ejecutar el código Cirq. Entre ellas se encuentran IonQ, Pasqal, Rigetti y AQT, sin embargo algunas de ellas no ofrecen servicio al público general o solo disponen de simuladores.

Figura 2.4: Logo de Cirq

2.3.3. Amazon Braket

Es un servicio de computación cuántica que ofrece la empresa Amazon para ayudar a investigadores a iniciarse en esta tecnología. Ofrece un entorno de desarrollo para explorar y desarrollar algoritmos cuánticos, simularlos con circuitos y ejecutarlos en hardware cuántico real. Braket ofrece una serie de herramientas para desarrolladores braket [30]. Una de ellas es el propio marco de desarrollo que es independiente del hardware, consiguiendo así simplificar el proceso de desarrollo y mejorar la experiencia. Además, como IBM, se puede optar por realizar los experimentos en blocs de notas de Jupyter administrados en la nube. En estos blocs de notas vienen preinstalados ciertos paquetes para empezar sin mucho esfuerzo y facilitan tutoriales y algunos algoritmos prediseñados.

Sobre el precio, Braket está incluido dentro de la parte gratuita (durante 12 meses) de Amazon Web Services. Sin embargo, para lanzar los trabajos en ciertos simuladores y en los sistemas hardware avanzados sí que existe una tarifa donde se paga en función del tiempo y el procesador [14]. En referencia a estos sistemas hardware, Amazon tiene como proveedores las empresas IonQ (iones atrapados) y Rigetti (superconductores) como equipos basados en puertas o D-Wave como solución al temple cuántico. Con esto, la empresa ofrece un abanico de posibilidades donde permite comparar las ventajas y desventajas de las diferentes tecnologías para cada aplicación.

2.4 Próximos pasos

A continuación se exponen los siguientes pasos y mencionan algunos factores limitantes o problemas que existen hoy en día en relación a la computación cuántica.

Existe una necesidad de investigación con el fin de reducir la diferencia que hay entre un posible ordenador cuántico útil (por ejemplo, para resolver problemas químicos complejos) y lo que se puede construir y probar actualmente. Para ser más concretos, se busca ejecutar algoritmos útiles en un tiempo práctico sobre un ordenador cuántico real y confiable. El ritmo de desarrollo actualmente es alto y hay investigadores diseñando lenguajes de programación, experimentando con distintas técnicas de construcción, escalabilidad y arquitecturas para lograr cerrar el *gap* o hueco que existe. A continuación se

mencionarán algunas de las recomendaciones de investigación propuestas en *Next Steps in Quantum Computing:Computer Sciences Role* [16].

- Hay una necesidad básica de diseño de algoritmos que aprovechen la limitación de número de qubits y su precisión.
- Es prioridad una rama de investigación sobre sistemas criptográficos de clave pública post-cuánticos que puedan resistir ataques cuánticos, pese a que todavía no se pueda ejecutar el algoritmo de Shor con grandes claves.

Por otra parte, existen algunos factores limitantes a los que se enfrentan los investigadores actualmente:

- Diseño de algoritmos. Actualmente no existen muchos algoritmos cuánticos que aprovechen el número limitado de cúbits (y su escasa precisión) al máximo. Es por ello que no existen muchos problemas reales que se puedan abordar con lo desarrollado hasta ahora.
- Calidad de los cúbits. Debido a la decoherencia, la información con la que se hacen cálculos se pierde muy pronto. Si se consigue mejorar la calidad de los cúbits la información será mantenida más tiempo y se abrirá la posibilidad de ejecutar operaciones más complejas.
- Índice de errores. Es necesario desarrollar un sistema de corrección de errores cuanto antes debido al número creciente de cúbits y por lo tanto lo complicado que se vuelve mantener la integridad del estado cuántico.
- Sistemas de criptográficos post-cuánticos de clave pública. Es prioridad buscar buscar la manera de resistir a ataques cuánticos debido a la mejora exponencial de algunos algoritmos desarrollados. Además, se debe buscar una manera de distribuir la clave con protocolos que usen los efectos de la cuántica.

A pesar de todo esto, se estima que en 2030 el mercado de la computación cuántica llegue a aproximadamente 65 mil millones de dólares frente a los 500 millones en 2019 [20].

2.5 Argumentos contra los ordenadores cuánticos

Gil Kalai, matemático en la Universidad Hebrea de Jerusalén, Israel, es uno de los personajes más destacados de un grupo de científicos que piensan que la computación cuántica no es posible que sea computacionalmente superior. En una entrevista para Quantamagazine, Kalai indica que limitar el ruido de un computador cuántico también limitaría la potencia de cómputo de este. Sobre la corrección de error cuántico asegura que se necesitarían 100 o incluso 500 cúbits físicos para conseguir un solo cúbit lógico de alta calidad [15].

En respuesta a un artículo publicado por Nature, prestigiosa revista científica, donde se describía un experimento con un computador de 53 cúbits de Google [13], Kalai publicó otro documento a modo de respuesta. En el primero, la multinacional afirmó que se lograba la ventaja cuántica con un ordenador Noisy Intermediate-Scale Quantum (NISQ) que, supuestamente, solucionan parcialmente los problemas de ruido y decoherencia. Sin embargo, Kalai afirma que los circuitos NISQ son básicamente dispositivos

clásicos de bajo nivel [3]. Por esto, según su naturaleza, los sistemas NISQ no soportan las ventajas de la cuántica. Es decir que el índice de ruido no se puede reducir al nivel que permita una ventaja cuántica. Para conseguir una corrección de errores cuánticos de buena calidad es necesario un nivel de ruido inferior al que se requiere para lograr la ventaja cuántica. Otro argumento que utiliza es que pese a que para la corrección de error cuántico se necesita una alta fidelidad para decenas o centenas de qubits, la ventaja cuántica se puede demostrar con poca fidelidad (base en la que se apoya el experimento de Google). El matemático concluye negando la posibilidad de una computación cuántica de gran escala basada en la corrección de errores cuánticos [3].

2.6 Conceptos relacionados con la computación cuántica

A continuación se introducen algunos de los conceptos técnicos que se usan a lo largo del trabajo y los necesarios para entenderlos. El objetivo es poner en contexto al lector para así entender cómo se ha logrado llegar a la solución. Se recomienda para todo aquel que desee profundizar en los conceptos, tanto de computación cuántica como de mecánica cuántica, el capítulo *2. Bits and Qubits* del libro A Primer on Quantum Computing [18] al cual se cita en varias ocasiones en esta sección. En el glosario, 7, también se recogen términos relacionados.

2.6.1. Computación reversible

Hace referencia a la parte de la computación que permite obtener el valor de entrada dados el valor de salida y la operación. Las operaciones que permiten las permutabilidad son reversibles; en el ámbito de las operaciones con un solo bit clásico existen la identidad y la negación. Sin embargo, el valor 0 (constante 0) y constante 1 son irreversibles. Los ordenadores cuánticos solo emplean operaciones reversibles. La evolución de los estados cuánticos se rige por la propiedad de unitariedad de la física cuántica. Esto significa que toda operación sobre un estado cuántico debe mantener la suma de las probabilidades a 1. Toda puerta lógica cuántica se deberá implementar como un operador unitario, por lo tanto será reversible.

2.6.2. Estado cuántico

El estado cuántico es un término matemático y abstracto y se trata de la descripción del estado físico de un sistema cuántico. Sin embargo, el estado cúantico no es el estado en el que se puede encontrar, ya que siempre que se observa un objeto cuántico se obtiene un valor específico.

2.6.3. Notación Bra-ket

La notación bra-ket [18], o notación Dirac en honor a su creador, es una manera matemática de describir estados cuánticos. Un *ket* $|\psi\rangle$ es un simple vector y *bra* $\langle\psi|$ es la matriz adjunta del vector y cuando se juntan forman el *braket* $\langle\psi\rangle$ o producto interior. Esta es la convención empleada para describir los estados cuánticos que codifican los valores cero y uno:

$$\begin{bmatrix}1\\0\end{bmatrix} = |0\rangle\,, \begin{bmatrix}0\\1\end{bmatrix} = |1\rangle \tag{2.1}$$

2.6.4. Producto tensorial

El producto tensorial se expresa con $\otimes$ y consiste en una operación sobre dos matrices de tamaño indefinido. Es la operación que describe sistemas compuestos por múltiples subsistemas y cada subsistema se compone por un espacio vectorial, espacio de Hilbert. Por lo tanto, los vectores $|\phi_I\rangle$ y $|\psi_{II}\rangle$ describirían los estados del sistema I y II con el estado del sistema completo dado el producto tensorial se puede representar $|\phi\rangle \otimes |\psi\rangle$, siendo ϕ, ψ dos vectores, pudiendose simplificar en $|\phi\psi\rangle$.

Un ejemplo de producto tensorial para dos vectores $|\phi\rangle, |\psi\rangle \in C^2 es$

$$\begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} \otimes \begin{pmatrix} \psi_1 \\ \psi_2 \end{pmatrix} = \begin{pmatrix} \phi_1\psi_1 \\ \phi_1\psi_2 \\ \phi_2\psi_1 \\ \phi_2\psi_2 \end{pmatrix} \tag{2.2}$$

Junto a la notación braket, así es como se representaría el estado de dos cúbits inicializados a 0:

$$|0\rangle \otimes |0\rangle = |00\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.3}$$

2.6.5. Cúbit

Un cúbit [35] es, desde el punto de vista físico, una partícula subatómica como el electrón o el fotón. El objetivo es aislarlos de manera que sean controlables en un estado cuántico. A nivel matemático, un cúbit se representa con $\begin{pmatrix} a \\ b \end{pmatrix}$ donde a y b son números complejos y que $||a||^2 + ||b||^2 = 1$.

Estos cúbits poseen propiedades como la superposición que permite estar simultáneamente en varios estados consiguiendo así representar más combinaciones posibles en un periodo de tiempo que un bit clásico. Cuando se mide el cúbit se dice que colapsa a 0 o 1 y esto se suele hacer al final de la computación para obtener los resultados. Si un cúbit tiene un valor $\begin{pmatrix} a \\ b \end{pmatrix}$ colapsará a 0 con una probabilidad de $||a||^2$ y a 1 con probabilidad de $||b||^2$.

Otro término importante directamente relacionado con el cúbit es el de entrelazamiento cuántico: esto aparece cuando los dos miembros de un par de partículas (o cúbits) existen en un mismo estado cuántico. Es decir que cualquier cambio de estado de una partícula afectará de manera inmediata a la otra sinimportar la distancia que las separe. Desde el punto de vista matemático, se dice que dos cúbits están entrelazados si el producto de sus estados no se puede factorizar:

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} c \\ d \end{pmatrix}$$

$$ac = \frac{1}{\sqrt{2}}, \quad ad = 0, \quad bc = 0, \quad bd = \frac{1}{\sqrt{2}}$$

Finalmente, la decoherencia cuántica es la pérdida de la pureza del comportamiento cuántico al interactuar con el entorno que sufren los cúbits [34]. El ruido, es decir cualquier alteración en la temperatura o vibración de un computador cuántico, es el principal enemigo de la cuántica. La decoherencia provoca la pérdida de la información e induce a errores en los cálculos, problema que se intenta solucionar con algoritmos y más cúbits que reajusten los cálculos. Esto tiene una limitación y es que para conseguir un cúbit lógico, o altamente fiable, se necesitan muchos cúbits y este es uno de los puntos mencionados en el apartado 2.5.

CAPÍTULO 3
Análisis del problema

3.1 Especificación de requisitos

En los siguientes subapartados se puede ver la especificación de requisitos realizada de acuerdo a técnicas y métodos vistos en asignaturas como Análisis y Especificación de Requisitos.

3.1.1. Identificación de Stakeholders

Los Stakeholders son las partes interesadas e involucradas en el sistema de manera directa o indirecta. Para este producto, se han detectado los siguientes:

- Usuario.
- Servicios de ejecución de circuitos cuánticos en hardware real.
- Centros docentes.

3.1.2. Perfiles de Stakeholders

En la tabla 5.1 se muestran los distintos stakeholders indicando si son usuarios directos o no y una breve descripción de los intereses en el proyecto.

Nombre	Usuario directo (Sí/No)	Intereses
Usuario	Sí	Aprender e investigar sobre la computación cuántica
Servicios de hardware cuántico real	No	Recibir más ejecuciones a través del producto, creciendo así su popularidad y sus usuarios.
Centros docentes	No	Mejorar la enseñanza a través de la aplicación para mantener satisfechos a los alumnos.

Tabla 3.1: Tabla con los perfiles de los stakeholders

3.1.3. Modelo de Dominio

El diagrama de modelo de dominio tiene como objetivo representar el vocabulario y las relaciones que existen entre los distintos conceptos del dominio. Con esto se consigue establecer un lenguaje común entre todas las partes involucradas en el proyecto.

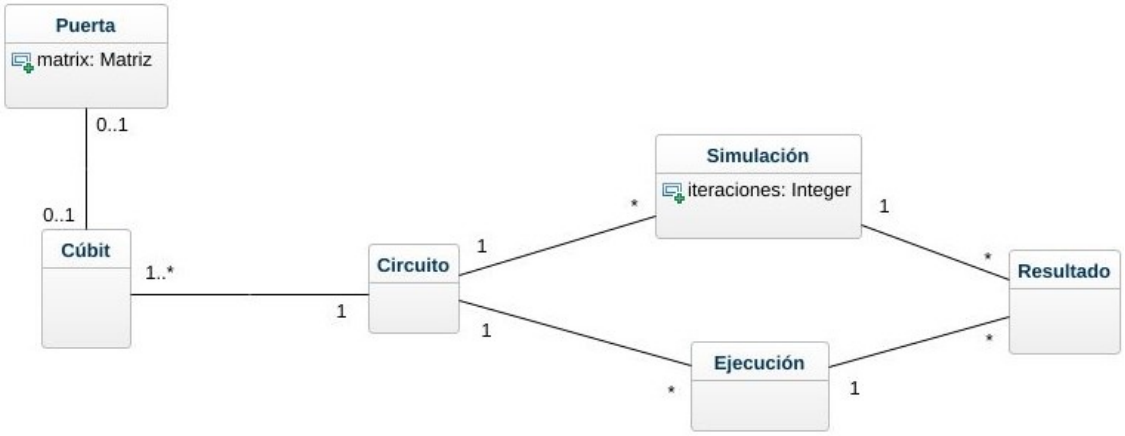


Figura 3.1: Diagrama del modelo de dominio

3.1.4. Diagrama de Contexto

El diagrama de contexto muestra, a alto nivel, el sistema, sus límites y las entradas y salidas. Las entradas y salidas, también llamadas entidades, son tantos los actores como los sistemas externos con los que el sistema interactúa.

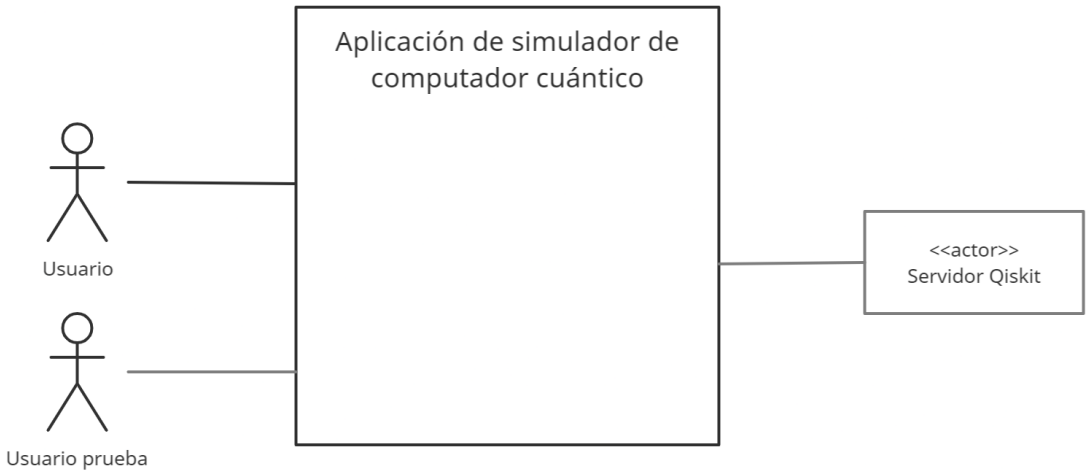


Figura 3.2: Diagrama de contexto

3.1.5. Casos de Uso

Para analizar un sistema es necesario tener claro cómo se comunican los distintos usuarios o subsistemas y el comportamiento del propio sistema. Es por ello que se ha decidido presentar esta información mediante un diagrama de casos de uso y una descripción en forma de tabla. Se han agrupado por características.

3.1.6. Característica 1 - Puertas

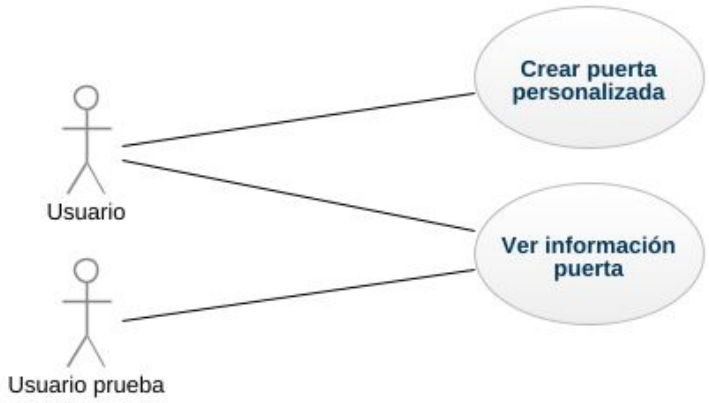


Figura 3.3: Diagrama de casos de uso de la característica Puertas

Referencia:	**CU1**
Nombre:	Crear puerta personalizada
Descripción:	El usuario podrá crear su propia puerta lógica a partir de una matriz.
Actor:	Usuario
Relaciones:	-
Precondición:	-
Comentarios:	Solo lo podrá hacer el usuario con licencia.

Tabla 3.2: Caso de Uso 1

Referencia:	**CU2**
Nombre:	Ver información de la puerta
Descripción:	El usuario podrá ver información sobre la puerta al clicar sobre ella
Actor:	Usuario, Usuario prueba
Relaciones:	-
Precondición:	-
Comentarios:	-

Tabla 3.3: Caso de Uso 2

3.1.7. Característica 2 - Circuitos

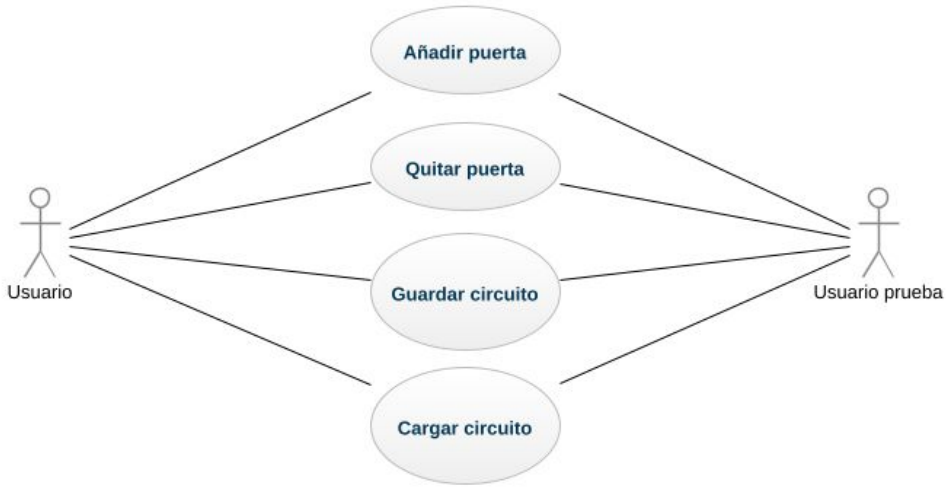


Figura 3.4: Diagrama de casos de uso de la característica Circuitos

Referencia:	**CU3**
Nombre:	Añadir puerta al circuito
Descripción:	El usuario añadirá una puerta al circuito arrastrándola al cúbit que desee.
Actor:	Usuario, Usuario prueba
Relaciones:	-
Precondición:	-
Comentarios:	-

Tabla 3.4: Caso de Uso 3

Referencia:	**CU4**
Nombre:	Quitar puerta del circuito
Descripción:	El usuario quitará una puerta del circuito arrastrándola a la basura.
Actor:	Usuario, Usuario prueba
Relaciones:	-
Precondición:	Tiene que haber al menos una puerta sobre el circuito. (CU3)
Comentarios:	-

Tabla 3.5: Caso de Uso 4

Referencia:	**CU5**
Nombre:	Guardar circuito
Descripción:	El usuario tendrá la opción de guardarse el circuito en un fichero de manera local.
Actor:	Usuario, Usuario prueba
Relaciones:	-
Precondición:	-
Comentarios:	-

Tabla 3.6: Caso de Uso 5

Referencia:	CU6
Nombre:	Cargar circuito
Descripción:	El usuario tendrá la opción de cargar el circuito desde un fichero de manera local.
Actor:	Usuario, Usuario prueba
Relaciones:	-
Precondición:	Previamente se debe de haber guardado un circuito (CU5)
Comentarios:	-

Tabla 3.7: Caso de Uso 6

3.1.8. Característica 3 - Simulación y ejecución

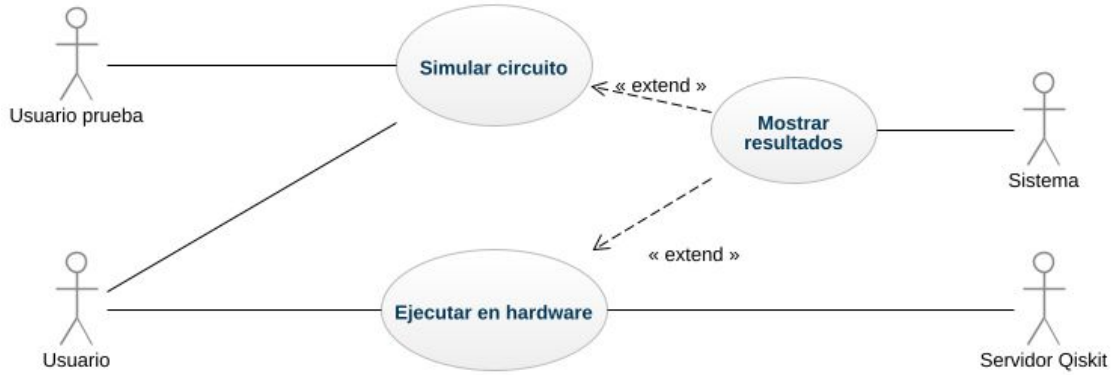


Figura 3.5: Diagrama de casos de uso de la característica Simulación y ejecución

Referencia:	CU7
Nombre:	Simular circuito
Descripción:	El usuario puede simular el circuito previamente diseñado.
Actor:	Usuario, Usuario prueba
Relaciones:	-
Precondición:	Se debe de haber creado un circuito.
Comentarios:	-

Tabla 3.8: Caso de Uso 7

Referencia:	CU8
Nombre:	Ejecutar circuito en hardware
Descripción:	El usuario manda el circuito a un servidor externo donde se ejecuta el circuito y este devuelve los resultados.
Actor:	Servidor Qiskit
Relaciones:	-
Precondición:	Se debe de haber creado un circuito.
Comentarios:	-

Tabla 3.9: Caso de Uso 8

Referencia:	CU9
Nombre:	Mostrar resultados
Descripción:	Se muestran los resultados una vez finalizada la simulación.
Actor:	Sistema
Relaciones:	CU7, CU8
Precondición:	Se debe de haber creado un circuito y simulado o ejecutado sin errores.
Comentarios:	En caso de algún error en la simulación o ejecución se mostrará un mensaje de error en la misma pantalla de resultados.

Tabla 3.10: Caso de Uso 9

3.2 Análisis y matriz DAFO

Con el fin de evaluar la viabilidad de la aplicación y definir la estrategia a seguir se ha realizado un análisis DAFO: se trata de un gráfico que muestra las debilidades, amenazas, fortalezas y oportunidades del mercado donde se posiciona la aplicación a desarrollar. Las fortalezas y debilidades son aspectos internos que sí se pueden impulsar y corregir, respectivamente. Sin embargo, las oportunidades y amenazas son aspectos externos que no es posible controlar pero sí tener en cuenta para actuar en consecuencia.

Figura 3.6: Matriz DAFO

Como se puede observar en la matriz, existen unas fortalezas muy claras que pueden hacer un hueco en el mercado. Las debilidades y, sobretodo, las amenazas se tendrán que solucionar o preparar un plan de actuación. Finalmente, de las oportunidades es importante destacar el ritmo al que está creciendo el interés en este campo, lo que facilitará la llegada de personas curiosas a nuestra aplicación.

3.3 Modelo de negocio

La aplicación a desarrollar está disponible inicialmente para ordenador, concretamente para el sistema operativo Windows. Además, se trata de una aplicación de carácter educativo, destinada tanto a particulares como a instituciones de educación. Teniendo en cuenta esto, y las múltiples opciones de las que se pueden generar ingresos a través de

una aplicación, se ha decidido optar por un modelo de negocio tradicional donde habrá un pago único por licencia. Con esta licencia se obtendrá un acceso total a la aplicación.

Es un modelo de negocio poco arriesgado donde se mitiga el problema del desconocimiento del cliente sobre el producto ofreciendo una versión de prueba gratuita de 7 días de duración con funcionalidad reducida. Para ello será necesario ingresar los datos bancarios pudiéndose cancelar el cobro de la licencia en cualquier momento antes de finalizarse el periodo de prueba. Con esta versión de prueba se limitará el número de cúbits a 3 y el número de simulaciones a 10 diarias además de ciertas funcionalidades como crear puertas personalizadas.

A continuación se analiza el apartado económico, la previsión de ingresos y gastos para los próximos dos años divididos por trimestres. El precio de preventa de una licencia individual es 19.95€. El importe que se tendrá que pagar por licencia de un dispositivo a partir del día oficial de lanzamiento es 24.95€y 22.95€por licencia para equipos de entre 5 y 10 dispositivos. Para grupos más grandes será necesario contactar por correo electrónico para conocer más detalles sobre el uso que se le vaya a dar (comercial, derechos de autor, etc.). Con estos datos, se propone como objetivo para la finalización del cuarto trimestre del segundo año haber superado la cifra de 700 licencias vendidas en total.

Tipo Licencia	Precio Licencia	Preventa	Año 1. Tr. 1	Año 1. Tr 2	Año 1. Tr 3	Año 1. Tr 4	Año 2. Tr 1	Año 2. Tr 2	Año 2. Tr 3	Año 2. Tr 4	TOTAL	TOTAL Precio
Licencia preventa	19.95 €	10	0	0	0	0	0	0	0	0	10	199.50 €
Licencia individual	24.95 €	0	15	15	20	25	30	50	70	90	315	7.859.25 €
Licencia grupo 5-10	22.95 €	0	7	10	15	15	20	25	30	65	187	4.291.65 €
Licencia Grupo Numeroso 1 (11-20 dispositivos)	20.00 €	0	0	0	0	12	15	14	16	20	77	1.540.00 €
Licencia Grupo Numeroso 2 (>20 dispositivos)	18.50 €	0	0	0	0	0	25	35	50	65	175	3.237.50 €
											764	17.127.90 €

Figura 3.7: Proyección de ingresos de licencias por trimestre durante los primeros dos años

En la figura 3.7 se puede observar datos como que se estima que se venderán 10 licencias durante la preventa o que en el cuarto trimestre del segundo año habrá un grupo numeroso interesado en obtener 64 licencias, por lo que se le rebaja el precio de la licencia a 18.50€la unidad. En total se estima que se venderán alrededor de 649 licencias (1 licencia por dispositivo) con un valor aproximado de 14.500€.

En cuanto a los gastos de producción, considerando que una primera versión será realizada por un solo desarrollador, serán bajos. Los gastos en marketing serán un punto importante para la difusión de la aplicación a través de diversos canales (medios especializados, revistas tecnológicas y divulgadores científicos). Siempre habrá un gasto fijo en cuanto a publicidad, sin embargo se realizarán campañas promocionales cuando se considere oportuno y en función de la actualidad sobre el mundo cuántico. Finalmente, se estima que existe un coste de mantenimiento de la web oficial cuando se lance al mercado.

Gastos	Preventa	Año 1. Tr. 1	Año 1. Tr 2	Año 1. Tr 3	Año 1. Tr 4	Año 2. Tr 1	Año 2. Tr 2	Año 2. Tr 3	Año 2. Tr 4	TOTAL
Desarrollador 1	1.600 €	1.600 €	1.600 €	1.600 €	1.600 €	1.600 €	1.600 €	1.600 €	1.600 €	14.400 €
Marketing	350 €	150 €	150 €	150 €	150 €	150 €	150 €	150 €	150 €	1.550 €
Mantenimiento de la web	80 €	80 €	80 €	80 €	80 €	80 €	80 €	80 €	80 €	720 €
Ordenador	600 €									600 €
										17.270 €

Figura 3.8: Proyección de gastos por trimestre durante los primeros dos años

Observando ambas tablas se puede concluir fácilmente que, aproximadamente a partir del tercer año, se obtienen beneficios del proyecto.

3.4 Lean Canvas

El *Lean Canvas* [33] es un lienzo de modelos de negocio que recoge de manera muy visual la información necesaria para diseñar un modelo de negocio. Es una adaptación del lienzo *Business Model Canvas*. Los números de los bloques indican en qué orden se deben rellenar y a continuación se ofrece una breve descripción de cada uno de ellos:

1. Clientes. Identificar a los conjuntos de clientes a los que piensa ir destinado el producto. Establecer los *early adopters* que nos retroalimentarán con sus opiniones.
2. Problema. Indicar cuáles son las necesidades que satisface o problemas que resuelve el producto. Realizar un análisis de los competidores.
3. Proposición de valor. Mostrar con un mensaje breve y conciso lo diferencial que aporta tu producto.
4. Solución. Establecer las características clave de nuestra propuesta para enfocarse en ellas.
5. Canales. Determinar las vías por las que conocerán el producto los clientes.
6. Ingresos. Establecer el modelo de negocio, cómo se generan los ingresos.
7. Costo. Indicar los posibles costos que conlleva el producto propuesto.
8. Métricas. Definir las métricas para evaluar el valor y crecimiento del negocio.
9. Ventaja competitiva. Recoger los puntos más innovadores y exclusivos de la propuesta.

El *Lean Canvas* elaborado para este producto es el siguiente:

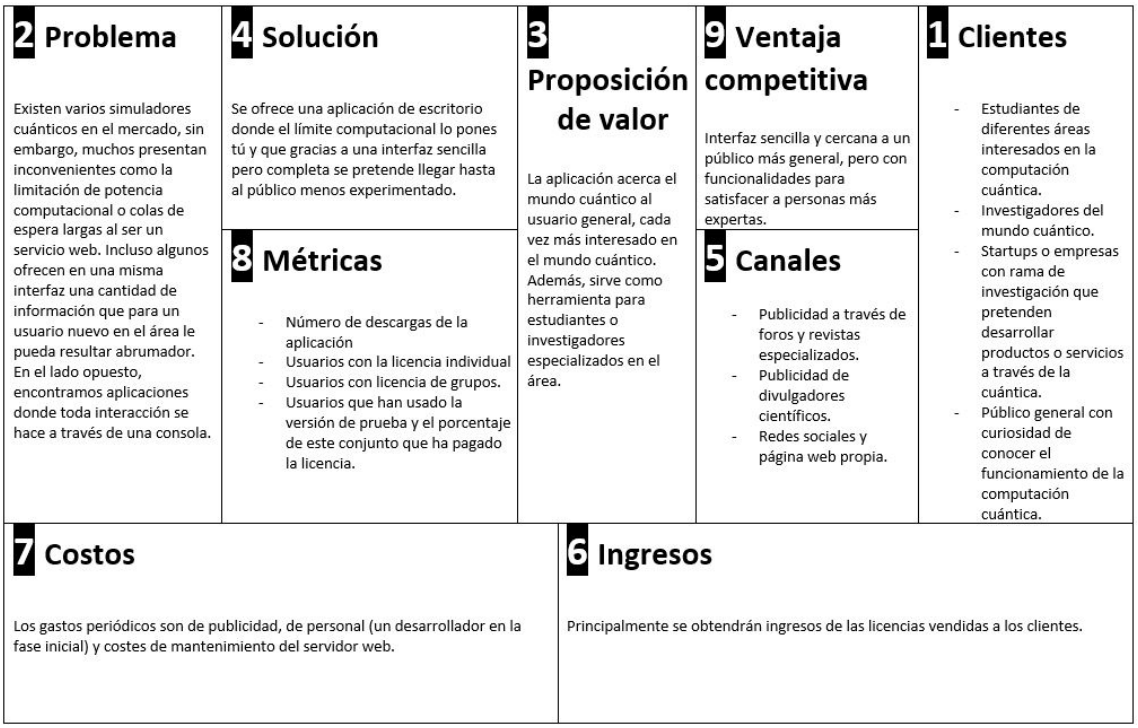


Figura 3.9: Lean Canvas sobre el producto

CAPÍTULO 4
Diseño de la solución

4.1 Arquitectura del Sistema

En general, existen varias soluciones al problema de cómo interactúan las diferentes partes del sistema. En este caso el patrón arquitectónico seguido, teniendo en cuenta la tecnología y la plataforma sobre la que se desarrolla, se trata una modificación del patrón Modelo-Vista-Controlador (MVC). Debido de nuevo a cómo está orientado el framework utilizado [8], se decide fusionar la vista y el controlador dando como resultado la arquitectura Modelo - Vista. De esta manera se mantiene la separación de cómo está almacenada la información y cómo se presenta al usuario y se fusiona la lógica con la vista.

- El modelo se comunica con la fuente de datos, ofreciendo una interfaz para el resto de componentes.

- La vista solicita la información al modelo y realiza la lógica necesaria (papel del controlador) y la muestra a través de los QWidget[19] (papel de la vista). También es el punto de entrada para interacciones con el usuario. Entre paréntesis se han indicado los papeles de una arquitectura MVC tradicional.

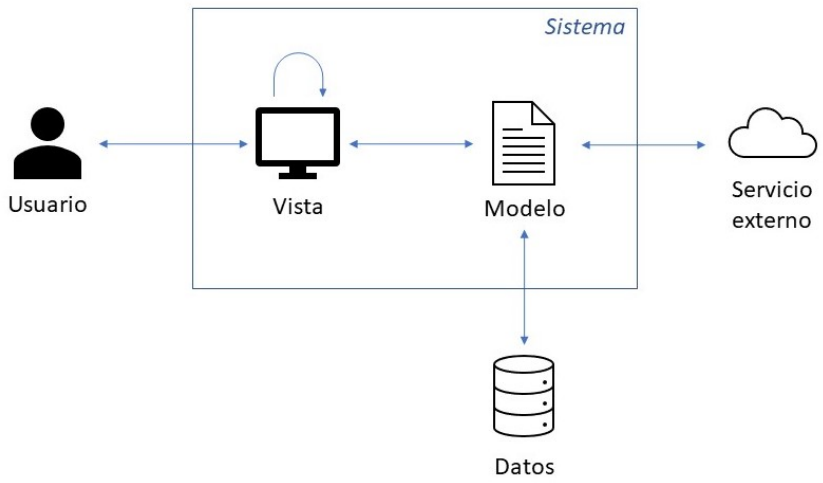


Figura 4.1: Esquema básico de la arquitectura utilizada en la aplicación.

4.2 Diseño detallado

En esta sección se detalla cómo se desgrana la arquitectura presentada anteriormente a nivel de módulos y clases y se describen las relaciones existentes.

4.2.1. Organización de clases

En primer lugar, el proyecto tiene cuatro subcarpetas correspondientes a los tres módulos detectados y una carpeta con imágenes. Estos tres módulos, cuyos nombres coinciden con las carpetas, son los siguientes:

Módulo *models*

En este módulo es donde se ubican las clases objeto, los modelos, siguiendo la arquitectura previamente definida Modelo - Vista. A continuación se enumeran los ficheros o clases que contiene.

- Gate.py
- Circuit.py
- Simulation.py
- Qbit.py

Como se puede ver en la figura 4.1 se relaciona con dos entidades fuera del sistema. La primera de ellas se trata de un servicio externo, que mediante su API se envía un circuito para su posterior ejecución. Por lo tanto, la clase específica que mantiene esta relación es la del circuito cuántico. Por otro lado, también se puede observar que el Modelo se comunica con los Datos: esto representa el guardado y cargado del estado de la aplicación en memoria. Se guarda en un fichero de manera local información como el circuito actual y las puertas lógicas personalizadas que haya definido el usuario. Posteriormente esta información se puede cargar en el simulador.

Módulo *views*

Aquí se encuentran los ficheros destinados a la vista y la lógica detrás de ella. En este módulo se encuentran tanto las clases de las ventanas de la interfaz de usuario, representadas con el sufijo *Window*, como componentes de estas interfaces, es decir, conjuntos de Widget con el fin de crear un Widget más complejo.

- newGateWindow.py
- resultsWindow.py
- mainWindow.py
- qbitWidget.py
- qLabelClickWidget.py

Módulo *Conn*

Este módulo está pensado para realizar las conexiones necesarias con los servicios externos que se utilizan en el proyecto.

- IBMQuantum.py

En la clase IBMQuantum se prepara lo necesario para establecer la conexión con los servidores de IBM y poder ejecutar el circuito en un ordenador cuántico. Aquí se introducen credenciales del usuario, se establece la comunicación, se lanzan los circuitos y se reciben los resultados.

Carpeta *Images*

Es una carpeta para almacenar las imágenes de las puertas prediseñadas y las generadas dinámicamente a partir de puertas personalizadas.

- H.jpg
- Y.jpg
- X.jpg
- Z.jpg
- ID.jpg
- control.jpg

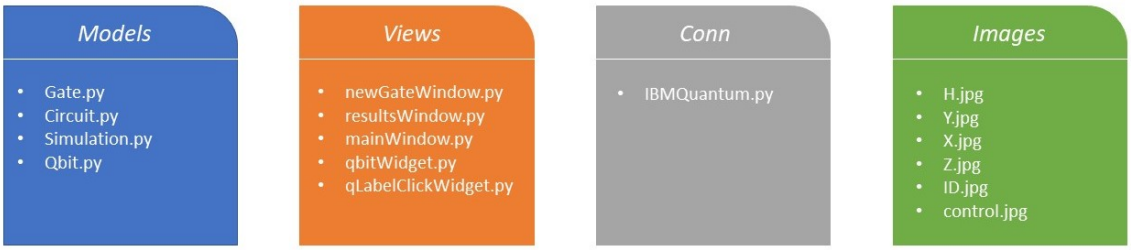


Figura 4.2: Agrupación de los ficheros por módulos

4.2.2. Diseño del fichero de guardado

Una de las funcionalidades de la aplicación es el guardado del circuito en el ordenador del usuario para su posterior carga en la aplicación. Para ello es necesario guardar la información contenida en el circuito (puertas lógicas y sus posiciones) en un esquema sencillo y sin pérdida de información. Se ha optado por almacenar las puertas en una lista de listas identificándolas por su atributo *id* (o un 1 en caso de ninguna puerta) de manera que la primera lista corresponde a las primeras n puertas siendo n el número de cúbits en el circuito. La primera lista contiene la primera columna de puertas, la primera puerta del cúbit 0, la primera puerta del cúbit 1,hasta la primera puerta del cúbit n. A continuación se ilustra con un ejemplo.

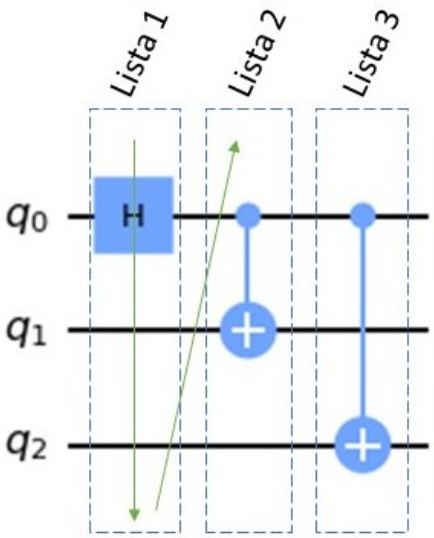


Figura 4.3: Ejemplo de circuito con indicaciones de la estructura del fichero de guardado

El circuito anterior es equivalente al siguiente esquema:

```
circuito = [
    [H,1,1],
    [control, X, 1],
    [control, 1, X]
  ]
```

Listing 4.1: Serialización del circuito para el guardado de ficheros

Finalmente, en este mismo archivo se incluye una entrada con las puertas lógicas que defina el usuario en la aplicación.

4.3 Tecnologías utilizadas

4.3.1. Python

Python [22] es un lenguaje multiparadigma de alto nivel muy popular en la actualidad debido a lo legible que es su código. Se desenvuelve muy bien en muchos ámbitos pero destaca en campos como desarrollo web, con entornos de trabajo como Django, en ciencia de datos y en inteligencia artificial.

En el proyecto se usará como lenguaje de backend para la aplicación de escritorio y como lenguaje base que utiliza el entorno de trabajo Qt.

Figura 4.4: Logo del lenguaje Python

4.3.2. PyQt

PyQt [23] es una adaptación de la biblioteca gráfica Qt al lenguaje Python. Con esta librería se pueden diseñar interfaces de usuario para aplicaciones de escritorio gracias a la herramienta Qt Designer. Además, ofrece la posibilidad de generar código desde Qt Designer y añadirle componentes de interfaz de usuario al diseñador escribiendo en lenguaje Python.

Figura 4.5: Logo del framework Qt

4.3.3. Visual Studio Code

Es un editor de código fuente gratuito [24] y de código abierto desarrollado por Microsoft. Algunos de sus puntos fuertes son que se trata de un editor muy personalizable y ofrece soporte a una gran cantidad de lenguajes. Esto es, en parte, gracias a las extensiones publicadas en la marketplace por su activa comunidad. También permite integrar Git y facilita la creación de terminales en el propio editor.

Figura 4.6: Logo del editor de texto Visual Studio Code

4.3.4. Github

Github [25] es una web y servicio en la nube que ofrece repositorios para gestionar y almacenar el código de un proyecto y para tener un registro y control de los cambios introducidos en el código. Utiliza en concreto el sistema de control de versiones Git.

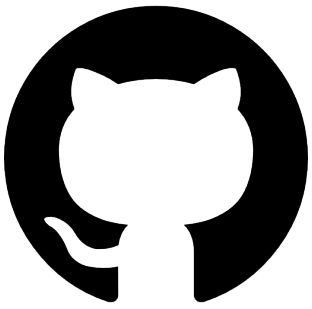


Figura 4.7: Logo de Github

4.3.5. Notion

Notion [26] es una herramienta de productividad creada por Notion Labs que se encuentra en auge y que ofrece varios componentes: notas, bases de datos, tableros kanban, calendarios, etcétera. Tiene muchas ventajas como son las plantillas creadas por la propia aplicación y por la comunidad o la facilidad para compartir tu espacio de trabajo con un compañero.

En esta ocasión se ha utilizado como herramienta de apoyo para la gestión del proyecto usando la plantilla personalizable de tablero Kanban. También se ha usado como organizador de bloc de notas rápidas sobre el trabajo.

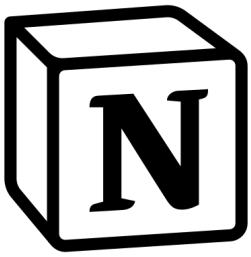


Figura 4.8: Logo de Notion

4.3.6. MockFlow

Mockflow [27] es una solución para crear nuestro propios *wireframes* en la nube. Un *wireframe* es un boceto donde se dibuja el esqueleto de la aplicación para organizar su diseño y sus elementos funcionales. Ofrecen un plan gratuito donde se pueden usar hasta 4 páginas. Existen elementos prediseñados que se pueden introducir directamente en el prototipo, como una barra de búsqueda de Google. Incluso los propios usuarios pueden subir a través de la tienda sus aportaciones. Está preparada para cualquier tipo de plataforma, móvil, tablet o ordenador y ofrecen distintos formatos para exportar el resultado.

Figura 4.9: Logo de MockFlow

4.3.7. GenMyModel

GenMyModel [28] es una herramienta de modelado en la nube. Permite crear muchos tipos de diagramas en UML o BPMN. Tiene un plan de pago por 25 dólares al mes por usuario y ofrece funcionalidades como colaborar a distancia sobre el mismo documento. La versión gratuita ofrece hasta 10 modelos privados y exportables en formatos como PDF o JPEG.

Figura 4.10: Logo de GenMyModel

4.3.8. Microsoft Project

Microsoft Project [29] es una herramienta para la gestión de proyectos para ayudar a administradores y jefes de proyectos. Ofrece funcionalidades como seguimiento de procesos, planificación temporal y de recursos o evaluar el ritmo de trabajo. Presenta esta en información en diversos gráficos como diagramas de Gantt. El software sigue las directrices y procedimientos que se describen en la Guía de los fundamentos para la dirección de proyectos (PMBOK).

Figura 4.11: Logo de Microsoft Project

4.4 Prototipos de la interfaz

Se han diseñado unos prototipos con el fin de validar el diseño de la interfaz de usuario antes del desarrollo. Los bocetos se han realizado con la herramienta MockFlow 4.3.6 y se muestran a continuación.

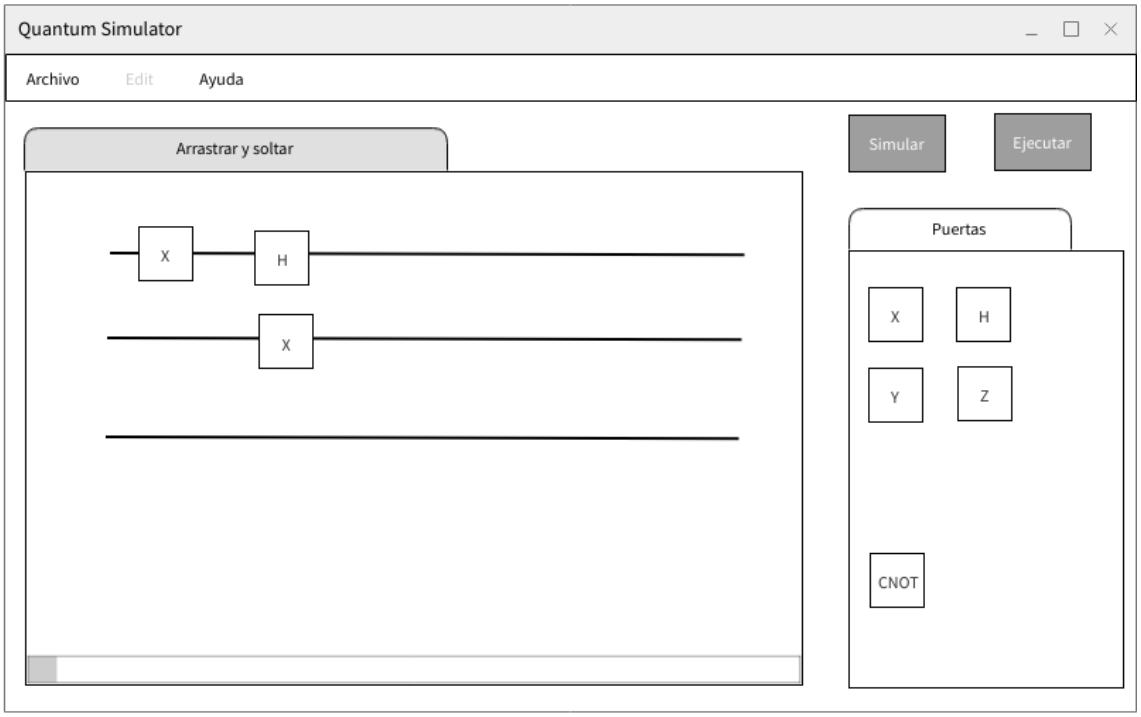


Figura 4.12: Ventana inicial de la aplicación

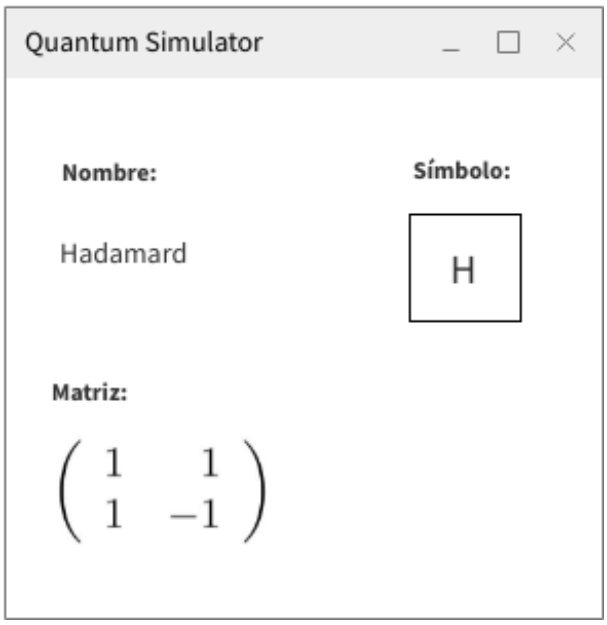


Figura 4.13: Ventana de información de la puerta

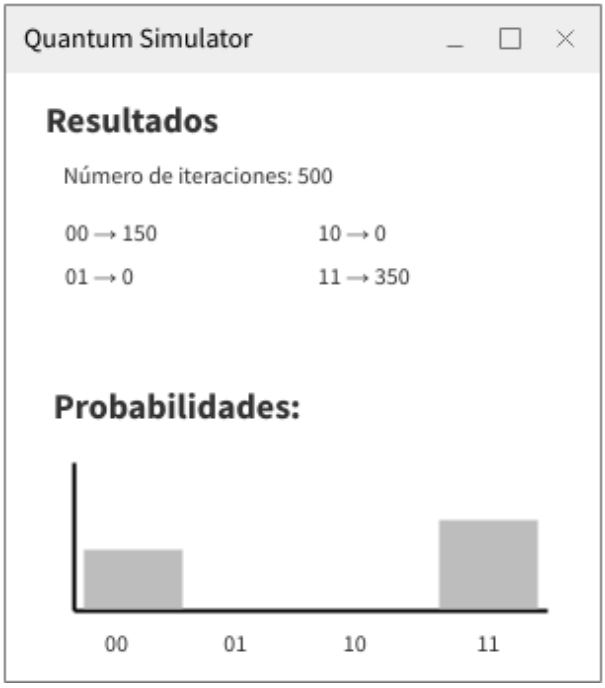


Figura 4.14: Ventana de resultados

CAPÍTULO 5
Desarrollo de la solución propuesta

5.1 Definición de los flujos de trabajo

Los flujos de trabajo seguidos para el desarrollo de la aplicación están adecuados a una metodología ágil. Un *workflow* o flujo de trabajo consiste en las etapas por las que pasara cada una de la porciones de trabajo o Unidad de Trabajo.

El primer workflow definido es uno ligeramente simplificado del utilizado en nuestro equipo para la asignatura de PIN. A continuación se muestra el flujo, que se aplica para tareas de programación:

Figura 5.1: Flujo de trabajo para las tareas de programación

Para tener una idea clara de lo que abarca en el flujo cada etapa se indica una definición breve de cada una:

- **Registrar**
  - Asegurarse de que la UT está en el flujo de trabajo correcto.
  - Comprobar que la UT no está repetido ni se solapa con otras.
  - Hacer una estimación preeliminar de Horas Ideales de Programación para la actividad del flujo **Programar**.
- **Especificar y diseñar**
  - Complementar añadiendo una descripción o algún documento adjunto para aportar más información. No solapar información con PA.
  - Elaborar mockups de la interfaz de usuario.
  - Definir las Pruebas de Aceptación para esta UT cumpliendo la plantilla establecida (Condiciones, Pasos, Resultado y Observaciones).
  - Una vez hecho lo anterior, revisar la estimación de Horas Ideales de Programación.
  - Indicar una prioridad para la tarea de manera que cuanto menor sea el número más prioridad tiene la UT.
  - Tener en cuenta una posible Refactorización y/o Aplicación de Patrones, indicándolo en la descripción de UT en caso de ser necesario.

- **Esperar Sprint**
  - Estado de las UT preparadas para ser desarrolladas en un Sprint. Así se consigue que no estén en programar sin que haya comenzado el sprint.
- **Programar**
  - Programar las interfaz, lógica y posible persistencia de los cambios asociados a la UT.
  - Realizar la Refactorización y/o Aplicación de Patrones en caso de haberse especificado.
  - Registrar el tiempo asociado a cada tarea.
- **Aplicar Pruebas de Aceptación**. Idealmente las realizaría una persona distinta a la que ha hecho la actividad Programar.
  - Aplicar las Pruebas de Aceptación marcando OK o KO en función de si se cumplen o no.
  - Si alguna prueba resulta ser KO, aportar información para reproducirlo en el campo Observaciones, finalizar la actividad Aplicar PA y devolver la actividad a Programar.
  - Si no hay ningún fallo en ninguna de las PA asociadas a la UT pasa a Terminar.
- **Terminar**. Estado de la UT que indica que se ha finalizado y no pasará por ninguna actividad más.

Para visualizar el flujo de trabajo y el estado de cada una de las tareas se ha utilizado como herramienta un tablero Kanban. La herramienta Notion proporciona un entorno donde poder gestionar el proyecto y a continuación se muestra el flujo de desarrollo y tareas de prueba para el desarrollo de las tareas de programación.

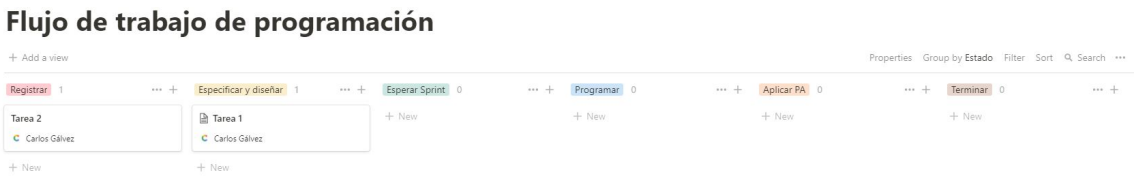


Figura 5.2: Tablero Kanban del flujo de trabajo para las tareas de programación

Figura 5.3: Ejemplo de tarea de programación y sus pruebas de aceptación

El flujo descrito anteriormente es para las tareas de programación, sin embargo existen otro tipo de tareas sin programación que seguirán el siguiente flujo de trabajo.

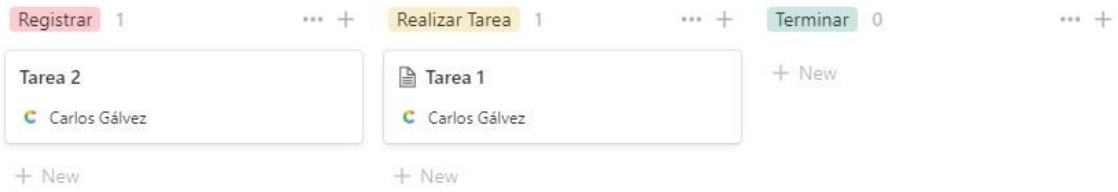


Figura 5.4: Tablero Kanban del flujo de trabajo para las tareas sin programación

Las actividades o fases por las que pasa cada Unidad de Trabajo son equivalentes a las del otro flujo y son autoexplicativas.

5.2 Planificación temporal

Se realiza una planificación temporal en base a las UT a implementar, su estimación de trabajo en Horas Ideales de Programación, la capacidad del equipo y el tiempo disponible. Para tener una visión general sobre las tareas a desarrollar, se ha realizado un mapa de características empleando la técnica de priorización MoSCoW.

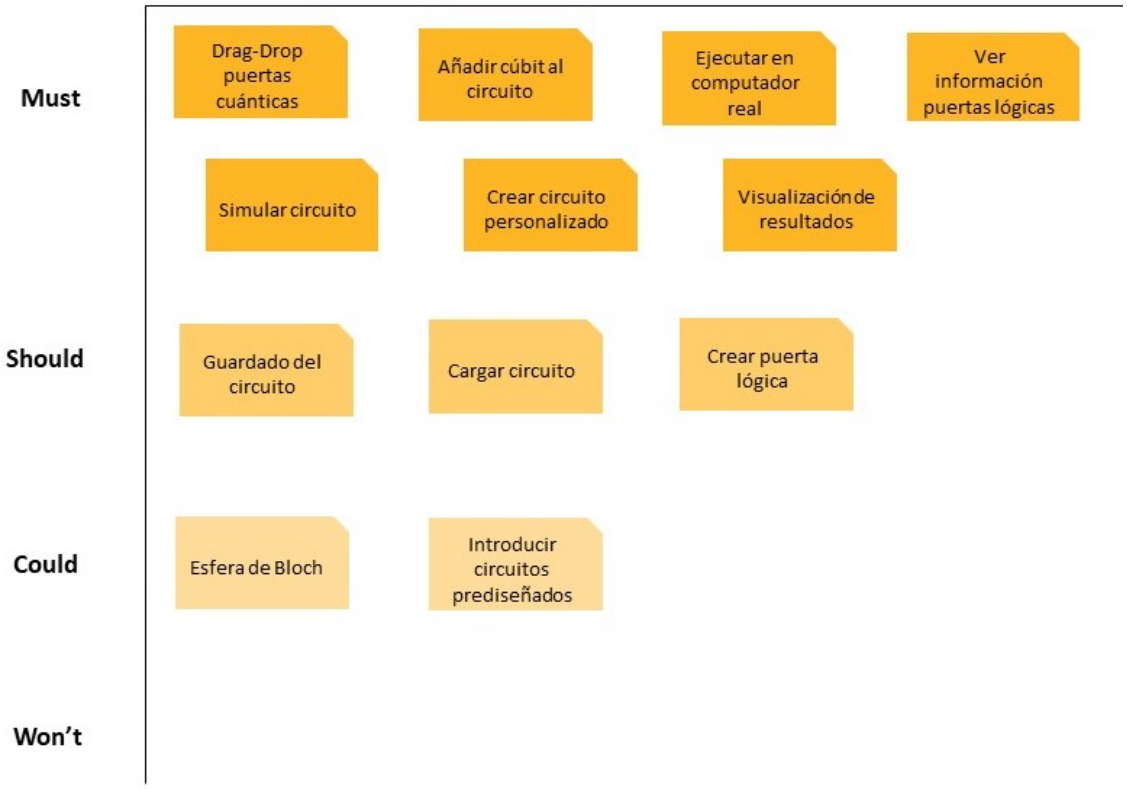


Figura 5.5: Mapa de características

Para gestionar los plazos de entrega, sobretodo en la parte del desarrollo de la aplicación, se utilizan *sprints*, núcleo de metodologías ágiles como Scrum. En este caso, cada ciclo de ejecución tiene una duración fija de dos semanas y un total de 3 *sprints* más un *Sprint* 0 de una semana que sirve de preparación y puesta a punto. Se intenta no modificar el alcance de cada uno de los *sprints*.

Para la gestión del alcance de cada sprint se ha tenido en cuenta las estimaciones realizadas con la técnica Horas Ideales de Programación (HIP). La capacidad de trabajo del equipo es de 22 horas a la semana. Por tanto, el alcance de cada *sprint* debe ser en torno a 22 * 2 (excepto el Sprint 0, 23). En las tablas 5.1 y 5.2 se pueden ver las tareas divididas por *sprints* con su estimación en horas y la total del *sprint*.

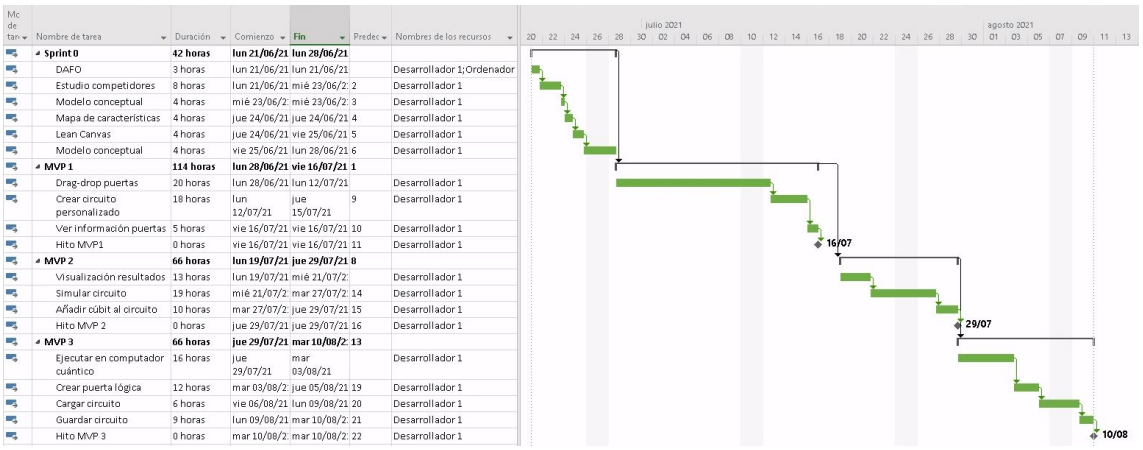
Sprint 0		Sprint 1	
Tareas	Horas	Tareas	Horas
DAFO	3	Drag-drop puertas	20
Mapa de características	4	Crear circuito personalizado	18
Estudio competidores	8	Ver información puertas	5
Lean Canvas	4		
Modelo conceptual	4		
Horas SP0: 23		Horas SP3: 43	

Tabla 5.1: Horas estimadas para cada tarea y el total de cada sprint

Sprint 2		Sprint 3	
Tareas	Horas	Tareas	Horas
Visualización resultados	13	Ejecutar en computador cuántico	16
Simular circuito	19	Crear puerta lógica	12
Añadir cúbit al circuito	10	Cargar circuito	6
		Guardar circuito	9
Horas SP2: 42		Horas SP3: 43	

Tabla 5.2: Horas estimadas para cada tarea y el total de cada sprint

Se ha realizado un diagrama de Gantt en Microsoft Project para programar y hacer un seguimiento del progreso del proyecto. Se han insertado los recursos necesarios, se ha configurado la fecha de inicio del proyecto para el 21 de junio y se ha definido un calendario personalizado 'Calendario TFG' que se ve reflejado al durar más de un día tareas de menos de 8 horas, que es el tiempo de jornada laboral por defecto en el programa. También se ha añadido al calendario un periodo vacacional que coincide con el desarrollo del MVP 1. Al haber un único recurso de trabajo se puede observar la secuencialidad de las tareas.



Nombre de tarea	Duración	Comienzo	Fin	Predec	Nombres de los recursos
Sprint 0	**42 horas**	**lun 21/06/21**	**lun 28/06/21**		
DAFO	3 horas	lun 21/06/21	lun 21/06/21		Desarrollador 1;Ordenador
Estudio competidores	8 horas	lun 21/06/21	mié 23/06/2	2	Desarrollador 1
Modelo conceptual	4 horas	mié 23/06/2	mié 23/06/2	3	Desarrollador 1
Mapa de características	4 horas	jue 24/06/21	jue 24/06/21	4	Desarrollador 1
Lean Canvas	4 horas	jue 24/06/21	vie 25/06/21	5	Desarrollador 1
Modelo conceptual	4 horas	vie 25/06/21	lun 28/06/21	6	Desarrollador 1
MVP 1	**114 horas**	**lun 28/06/21**	**vie 16/07/21**	1	
Drag-drop puertas	20 horas	lun 28/06/21	lun 12/07/21		Desarrollador 1
Crear circuito personalizado	18 horas	lun 12/07/21	jue 15/07/21	9	Desarrollador 1
Ver información puertas	5 horas	vie 16/07/21	vie 16/07/21	10	Desarrollador 1
Hito MVP1	0 horas	vie 16/07/21	vie 16/07/21	11	Desarrollador 1
MVP 2	**66 horas**	**lun 19/07/21**	**jue 29/07/21**	8	
Visualización resultados	13 horas	lun 19/07/21	mié 21/07/2		Desarrollador 1
Simular circuito	19 horas	mié 21/07/2	mar 27/07/2	14	Desarrollador 1
Añadir cúbit al circuito	10 horas	mar 27/07/2	jue 29/07/21	15	Desarrollador 1
Hito MVP 2	0 horas	jue 29/07/21	jue 29/07/21	16	Desarrollador 1
MVP 3	**66 horas**	**jue 29/07/21**	**mar 10/08/2**	13	
Ejecutar en computador cuántico	16 horas	jue 29/07/21	mar 03/08/21		Desarrollador 1
Crear puerta lógica	12 horas	mar 03/08/2	jue 05/08/21	19	Desarrollador 1
Cargar circuito	6 horas	vie 06/08/21	lun 09/08/21	20	Desarrollador 1
Guardar circuito	9 horas	lun 09/08/21	mar 10/08/2	21	Desarrollador 1
Hito MVP 3	0 horas	mar 10/08/2	mar 10/08/2	22	Desarrollador 1

Figura 5.6: Diagrama de Gantt con la planificación temporal del desarrollo de la aplicación

Nombre del recurso	Tipo	Etiqueta de material	Iniciales	Gi	Capacidad máxima	Tasa estándar	Tasa horas extra	Costo/U:	Acumular	Calendario base
Desarrollador 1	Trabajo		D		100%	6,00 €/hora	0,00 €/hora	0,00 €	Prorrateo	Calendario TFG
Marketing	Costo		M						Prorrateo	
Mantenimiento web	Costo		W						Prorrateo	
Ordenador	Material		O			600,00 €		0,00 €	Comienzo	

Figura 5.7: Hoja de recursos para el desarrollo de la aplicación

5.2.1. Producto mínimo viable

En el desarrollo de la idea de negocio se ha considerado interesante llevar a cabo el enfoque MVP, *Minimum Viable Prodcut* en inglés. El objetivo es tener un producto funcional con el fin de que los clientes puedan probarlo y validarlo. Por este motivo se ha dedicado cada *sprint*, período de tiempo, a un MVP.

MVP 1

El primer Producto Mínimo Viable tiene como objetivo principal desarrollar un simulador cuántico básico que opere con puertas simples. Estos circuitos personalizados, son introducidos en la aplicación arrastrando las puertas lógicas cuánticas a cada cúbit. Además, se consultará la información de la puerta a pulsar sobre ella.

Desarrollo MVP 1

A continuación podemos ver las UT asociadas al desarrollo de este primer MVP (que coincide con el primer *sprint*) y cómo se han organizado temporalmente:

A continuación se muestran los detalles de cada una de las tareas.

UT1: Drag-drop puertas cuánticas Se implementa la funcionalidad de arrastrar y soltar las puertas lógicas al circuito. Se trata, principalmente, del apartado gráfico y la lógica que esto conlleva.

Pruebas de aceptación:

- El usuario puede arrastrar la puerta que desee al circuito.
- El usuario puede eliminar una puerta del circuito arrastrándola desde el circuito a una zona determinada de borrado.
- Solo puede haber una puerta en una misma posición.

UT2: Crear circuito personalizado Se recoge la información dispuesta en la vista con el fin de crear el circuito diseñado previamente.

Pruebas de aceptación:

- El orden en el que las puertas se muestran se debe respetar.

UT3: Ver información de las puertas El usuario puede consultar la información asociada a las puertas definidas (y a las personalizadas, próximamente) al pulsar sobre ella.

Pruebas de aceptación:

- El usuario ve el nombre completo de la puerta.
- El usuario ve la matriz asociada de la puerta.
- El usuario ve el símbolo que representa a la puerta en el circuito.

MVP 2

En el segundo Producto Mínimo Viable se implementa la simulación del circuito creado en la misma aplicación. Además, se pueden visualizar los resultados de esta simulación. Finalmente, se prepara la aplicación para poder añadir nuevos cúbits al circuito sin afectar al funcionamiento.

Desarrollo MVP 2

UT4: Visualización de resultados El usuario puede ver los resultados de una simulación del circuito actual. Precondición: Haber simulado el circuito.

Pruebas de aceptación:

- El usuario ve un histograma con las probabilidades de cada posible estado.
- El usuario ve las probabilidades de los estados de cada cúbit.

UT5: Simular circuito El usuario puede simular de manera local el circuito diseñado previamente.

Pruebas de aceptación:

- Al pulsar en el botón Simular se inician los cálculos de la simulación.
- En caso de error en la simulación se muestra un mensaje en la pantalla.

UT6: Añadir cúbit al circuito El usuario puede añadir un cúbit nuevo al circuito.

Pruebas de aceptación:

- Al pulsar en el botón Añadir cúbit se crea un nuevo cúbit vacío en el circuito.

MVP 3

En este tercer *sprint* se busca tener desarrollado, junto a lo conseguido en los anteriores MVP, un producto con una alta riqueza en cuanto a funcionalidad. Se implementa la creación de tus propias puertas lógicas y la posibilidad de ejecutar el circuito en un sistema real a través de un servicio web. Como últimas funcionalidades, se le permite al usuario guardar el circuito de manera local y cargarlo.

Desarrollo MVP 3

UT7: Ejecutar en un computador cuántico real El usuario puede ejecutar el circuito diseñado y ejecutarlo en un computador cuántico a través de un servicio web.

Pruebas de aceptación:

- Al pulsar en el botón Ejecutar se envía la petición de ejecución al servidor.
- Al pulsar el botón se muestra en pantalla información del estado del trabajo (Pendiente, en proceso, finalizado)
- En caso de error en la ejecución, se muestra un mensaje por la pantalla.
- Una vez recibida la respuesta, se muestra la opción de ver resultados.

UT8: Crear puerta lógica El usuario es capaz de crear una puerta lógica personalizada.

Pruebas de aceptación:

- Al pulsar en el botón Añadir puerta se abre una ventana donde introducir los datos de la puerta.
- Serán obligatorios los datos: matriz, nombre y símbolo de la puerta.
- El símbolo de la puerta es en mayúsculas y como máximo se aceptan 4 letras [A-Z].
- Al pulsar en Aceptar la puerta se crea y se añade a la pestaña de Puertas Personalizadas.

UT9: Cargar circuito El usuario puede cargar un circuito desde un fichero previamente guardado en el dispositivo. Precondición: Haber guardado un circuito.

Pruebas de aceptación:

- Al pulsar en el botón Cargar se abre una ventana del explorador de archivos de Windows, de donde se selecciona el archivo.
- En caso de no ser un archivo correcto, notificar mediante una advertencia en pantalla.
- Al cargar el archivo correctamente se muestra el circuito en pantalla.
- Se habilita el atajo 'Control + L' para la carga del circuito.

UT10: Guardar circuito El usuario puede guardar un circuito en un fichero de manera local.

Pruebas de aceptación:

- Al pulsar en el botón Guardar se abre una ventana del explorador de archivos de Windows, donde se introduce el nombre del archivo.
- Al guardar el circuito correctamente se cierra el explorador de archivos.
- Se habilita el atajo 'Control + S' para guardado del circuito.

5.2.2. Hito final

Tras la consecución de los tres primeros MVP se considera que existe una primera versión de la aplicación lo suficientemente desarrollada como para considerarla una primera versión. Esta versión se recoge en el repositorio creado en Github para el proyecto como una *Release*. Allí se pone todos los archivos necesarios para su ejecución y las instrucciones correspondientes.

5.3 Implementación

En este apartado se detallará cómo se ha implementado la aplicación con las tecnologías utilizadas y los problemas o dificultades encontrados. Debido a lo extenso que podría ser el documento, solamente se mencionarán funcionalidades especialmente interesantes, ya sea por su complejidad o por su importancia a nivel del proyecto.

5.3.1. Aplicación en PyQt desarrollada

Se ha considerado importante describir como es la creación desde cero de una aplicación en PyQt para poder entender posteriores partes de la implementación.

Como ya se ha mencionado, PyQt es el framework que permite crear la GUI en Python. Como prerrequisito, Python deberá estar instalado en nuestro ordenador. Para instalarlo es tan sencillo como poner el comando `pip install PyQt5` en la consola CMD de Windows, sistema operativo utilizado para el desarrollo, pero es similar en otros sistemas operativos.

El código para iniciar una aplicación con un *Hola Mundo!* es el siguiente:

```
1 from PyQt5.QtWidgets import QApplication, QLabel
2 app = QApplication([])
3 label = QLabel("Hola ETSINF")
4 label.show()
5 app.exec_()
```

Listing 5.1: Aplicación sencilla en PyQt

En primer lugar se importan los paquetes necesarios, después se inicializa la aplicación con una **única instancia** de `QApplication`. A continuación creamos nuestro primer *widget* que consiste en un texto que se mostrará por pantalla al ejecutarse su método `show()`. Finalmente, con la última instrucción le decimos que ejecute la aplicación hasta que el usuario la cierre. Esta es la aplicación más básica que se puede hacer con PyQt.

Sin embargo, en la aplicación desarrollada se ha hecho uso de una gran variedad de *widgets* utilizando los *layouts* para disponer las piezas a nuestro gusto sobre la interfaz gráfica. Estos layouts se encargan también de redimensionar los QWidgets para ocupar todo el espacio posible en cada momento. También ofrecen, y se han usado en la aplicación desarrollada, señales y ranuras. Esta funcionalidad permite reaccionar a distintos eventos y ejecutar una parte del código determinada como efecto. Los eventos pueden ser desde que el usuario pulse en un botón hasta que se reciba una respuesta de una conexión a Internet.

Lógica del circuito en la interfaz

Una dificultad que surgió a la hora de la implementación de la interfaz fue decidir cual era el conjunto de *widgets* para disponer las puertas lógicas sobre una línea que represente el cúbit. Además, al arrastrar la puerta lógica al circuito se tiene que colocar en una posición específica, es decir, a la izquierda o a la derecha de alguna puerta previamente añadida en el cúbit. Para solucionar esto, se optó por utilizar un *QGridLayout* donde se disponen en forma de cuadrícula los distintos *QLabel*, que son las imágenes con el símbolo de la puerta. Los *QLabel* que son cuadrados con el borde a rayas indican que hay un hueco vacío donde colocar una puerta lógica. Para determinar la posición que ocupa la puerta que se arrastra al circuito se emplea la clase *QPoint*. Esta clase devuelve unas coordenadas con las que, al soltar la puerta en el cúbit, se comparan con el resto de puertas ya ubicadas para colocarla antes o después de estas. En el anexo A se puede consultar la imagen A.1 donde se ve lo anteriormente explicado.

Envío del circuito y obtención de resultados

Una vez hecho el circuito y seleccionado el ordenador donde se va a lanzar, se envía el trabajo. Aquí se pide que, a través del `job_monitor` notifique de en qué estado se encuentra: en cola, en ejecución o terminado. Finalmente obtenemos los resultados y se

muestran en la aplicación de manera similar a como si fuese una simulación en nuestra propia aplicación.

```
def run_circuit(self):
        self.job = execute(self.qc, backend=self.real_device)
        self.job_info = job_monitor(self.job)

def get_results(self):
        self.results = self.job.result()
        return self.results.get_counts(self.qc)
```

Listing 5.2: Ejecución del circuito y obtención de los resultados

5.3.2. Crear puerta personalizada

Otro punto interesante y sencillo de llevar a cabo es cómo se generan las puertas personalizadas. Desde la vista el usuario introduce los datos como el nombre de la puerta o la matriz que se guardan en el objeto sin ningún paso resaltable. En cambio, para poder representar la puerta en el circuito se necesita crear algún tipo de imagen a partir de lo que el usuario haya introducido en el campo Símbolo. Este problema no ocurre con las puertas prediseñadas porque se cargan a la hora de iniciarse la aplicación. Para solucionar este apartado se ha empleado una librería llamada *Pillow* [36] donde generamos una imagen con el texto introducido por el usuario y la guardamos en una carpeta junto al resto de imágenes de puertas lógicas. En el objeto *Gate*, que representa la puerta lógica, guardamos entonces la ruta donde se ubica esta nueva imagen.

```
H, L = (500,500)
img = Image.new('RGB', (H,L), color='white') #Se crea el objeto Image
d = ImageDraw.Draw(img)
w, h = d.textsize(gate.symbol, font=myFont)
d.text(((H-w)/2,(L-h)/2), gate.symbol, font=myFont, fill="black") #Se añade el
    texto en el centro
img.save(path) #Generamos la imagen en la ruta especificada
```

Listing 5.3: Generación de la imagen de la puerta personalizada

5.3.3. Ejecución del circuito en ordenador cuántico

Como ya se ha mencionado en el apartados anteriores, una de las funcionalidades de la aplicación era lanzar el circuito diseñado en un sistema de hardware cuántico. Para este trabajo se ha elegido usar Qiskit, SDK para trabajar con computadores cuánticos, en nuestro a caso a nivel de puertas y circuitos. Las API utilizadas son *Qiskit Terra* [37] y *Qiskit IBM Quantum (Provider)* [**?**]. A continuación se expone cual es el propósito de cada una y cómo se han utilizado e integrado en nuestra aplicación.

Creación del circuito en Qiskit

Partimos de un circuito creado y almacenado en nuestra aplicación con una estructura propia. Este circuito tendrá que ser transformado a una estructura capaz de ser interpretada por el ordenador cuántico sobre el que vamos a ejecutarlo. De esto se encarga *Qiskit Terra*. Es la base sobre lo que se sustenta todo el entorno Qiskit: circuitos, puertas, operadores, algoritmos, simuladores, herramientas, etc. A continuación mostramos las partes más importantes de este proceso:

```
def create_circuit(self, circuit : Circuit):
```

```
        qbits = len(circuit.serialized[0]) #Número de cúbits
        self.qc = QuantumCircuit(qbits, qbits) #Inicialización del circuito
        for column in circuit.serialized:
            curr_qbit = 0
            for item in column:     #Transformación de nuestras puertas a
                Qiskit
                if item.id == 'x':
                    self.qc.x(curr_qbit)
                elif item.id == 'y':
                    self.qc.y(curr_qbit)
                elif item.id =='z':
                    self.qc.z(curr_qbit)
                elif item.id == 'h':
                    self.qc.h(curr_qbit)
                elif item.id == 'c':
                    self.qc.control(curr_qbit)
            curr_qbit += 1 % qbits
        self.qc.measure_all()  #Añadimos una medición al final de cada cúbit
```

Listing 5.4: Transformación del circuito de la aplicación a código Qiskit.

Conexión a ordenador cuántico

Ahora realizamos la configuración previa para cargar las credenciales, que son obligatorias para lanzar el circuito en un dispositivo cuántico real. Seguidamente se elige el backend donde se ejecutará el circuito a través de la API *IBM Quantum* la parte de *Providers*.

```
IBMQ.load_account()
self.provider = IBMQ.get_provider(hub='ibm-q', group='open', project='main')
self.real_device = self.provider.get_backend('ibmq_santiago')
```

Listing 5.5: Selección del backend donde ejecutar el circuito

CAPÍTULO 6
Conclusiones

La temática del trabajo de final de grado propuesta por mi tutor considero que fue muy llamativa y novedosa. La realización de un simulador cuántico conlleva una investigación previa que me ha dado la oportunidad de conocer el campo de la computación cuántica e incluso ir más allá con la física relacionada. Esta parte de contínuo aprendizaje sobre las bases de la cuántica ha sido una de los aspectos que más ha hecho disfrutar y me ha abierto todo un mundo que desconocía y ahora me resulta muy atractivo. Gracias a este proyecto, a nivel técnico también he aprendido tecnologías desde cero que me han dado pie a continuar profundizando en ellas para lograr resultados profesionales.

Sobre la consecución de objetivos, ha sido satisfactoria a nivel general. Se ha puesto en conocimiento las bases de la computación cuántica y sus pecularidades y se han analizado los servicio más importantes de grandes empresas en el estado del arte. Se ha desarrollado una aplicación de escritorio con Python como lenguaje base y a través de ella se simulan circuitos cuánticos, se puede analizar los resultados de esta simulación y se lanzan circuitos en un servicio externo, en este caso en ordenador cuánticos de IBM Quantum. Finalmente se dedica todo un apartado para describir como se ha llevado a cabo la solución usando una metodología ágil.

Bien es cierto, que como se comenta en el apartado de trabajo futuro, la aplicación ha dejado algunos flecos y se tendrán en cuenta.

En relación a las dificultades aparecidas, en primer lugar me gustaría comentar el desconocimiento del campo de la computación cuántica. Gran parte del tiempo total empleado en este trabajo se ha destinado a la lectura de libros y artículos recomendados por mi tutor que me han resultado inmensamente útiles en los primeros pasos. Al ser una comunidad muy grande y joven no ha resultado difícil encontrar información y recursos por Internet, así que tirando del hilo iba encontrando nuevas fuentes de conocimiento. Sin embargo, a pesar de haber sido una dificultad notable, considero que ha sido lo que me ha hecho encontrar una nueva rama por la que quiero seguir investigando por cuenta propia.

Otra de las dificultades fue el aprendizaje de la tecnologías para la implementación de la aplicación de escritorio, sobretodo la parte de la interfaz gráfica. Con Python ya había trabajado previamente, pero mi nivel era muy básico. A eso había que sumarle la nula experiencia con el framework PyQt pero, al menos, el conocimiento que me dieron las prácticas de la asignatura Interfaces persona computador con JavaFX fue esencial para agilizar el aprendizaje en esta tecnología. Incluso el hecho de emplear librerías totalmente extendidas y usadas a nivel profesional como NumPy me reafirma en que todo lo aprendido me servirá en un futuro próximo.

Sin duda, personalmente pienso que las dificultades surgidas a lo largo del trabajo se han transformado en recompensas en forma de conocimiento.

6.1 Relación del trabajo desarrollado con los estudios cursados

A lo largo del grado se han cursado muchas asignaturas con las que se adquieren una gran cantidad de conocimientos tanto teóricos como prácticos. Este trabajo trata de demostrar que se han sabido aplicar correctamente todos estos años de aprendizaje.

En primer lugar, se ha utilizado en todo momento una metodología ágil, cosa que ha conseguido reducir los tiempos y dando una calidad aceptable al producto final. Esto ha sido posible gracias a las asignaturas Proceso de Software y Proyecto de Ingeniería de Software, donde se ha abordado la aplicación de las metodología ágil de una manera muy práctica. Junto a estas se conviene mencionar la asignatura Gestión de Proyectos, cursada en el tercer curso, ya que los conocimientos que me aportó se ven reflejados de manera directa e indirecta en la mayor parte del proyecto.

Análisis y Especificación de Requisitos es una asignatura totalmente necesaria para cualquier proyecto software. Se han seguido, en cierta medida, los pasos realizados en las prácticas de la asignatura para el apartado de análisis.

Otra asignatura que se considera que ha aportado mucho al proyecto es Mantenimiento y Evolución del Software. Siempre es importante recordar que del ciclo de vida del software, el mantenimiento es la parte más costosa por lo que tener presente esto puede ayudar. La mayoría de las acciones realizadas han sido parte del mantenimiento preventivo como reorganización del código para mejorar su legibilidad, respetar los nombres de las variables, métodos, etc. o añadir comentarios en partes del código importantes o complejas. Sin embargo, como se indica en la sección de trabajo a futuro, todavía queda camino por delante en este aspecto.

CAPÍTULO 7
Trabajo futuro

En cuanto al trabajo futuro sobre la aplicación desarrollada es recomendable cubrir unos flecos que han quedado por falta de tiempo, que son:

- Mejora de la interfaz gráfica del usuario a una más profesional. Cumplir las directrices de una aplicación de escritorio dependiendo del sistema operativo donde se lance para mejorar la experiencia de usuario [32].
- Refactorización del código para mejorar su mantenibilidad ayudando a los futuros desarrolladores que decidan aportar al proyecto.

Por otro lado también es interesante ampliar el alcance del proyecto ofreciendo nuevas funcionalidad que se han quedado fuera de esta primera versión, de nuevo, por falta de tiempo. Algunas de las funcionalidades que más valor aportarían al proyecto son las siguientes:

- Realizar un estudio económico más profundo y detallado sobre la viabilidad del negocio, así como posibles pivotajes en la plataforma o las fuentes de ingresos.
- Añadir la esfera de Bloch para visualizar geométricamente el estado del cúbit.
- Redactar documentación a nivel de usabilidad de la aplicación.
- Redactar documentación sobre los aspectos teóricos presentes en la aplicación para ayudar al usuario principiante (interpretar los resultados de la simulación, explicación de los algoritmos más importantes y cómo recrearlos, etc.)
- Añadir nuevos proveedores de ordenador cuánticos o *backends* donde ejecutar los circuitos.
- Crear una API REST para integrar nuestro simulador en otras aplicaciones y así extender su uso.

Bibliografía

[1] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences, (1985), 400(1818)*

[2] David Deutsch, Richard Jozsa. Rapid solution of problems by quantum computation *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences, (1992), 439(1907)*

[3] Gil Kalai. The Argument against Quantum Computers, the Quantum Laws of Nature, and Googles Supremacy Claims 2021

[4] Kok P., Munro W., Nemoto K., Ralph T., Dowling J., Milburn G. Linear optical quantum computing with photonic qubits *Reviews of Modern Physics, (2007), 79(1)*

[5] Huang, He-Liang; Wu, Dachao; Huang; Fan, Daojin; Zhu, Xiaobo. Superconducting quantum computing: a review *Science China Information Sciences, 2020, 63(8)*

[6] DiVincenzo, David P. The Physical Implementation of Quantum Computation. *Fortschritte der Physik, (2000), 48(9-11)*

[7] Shor, Peter W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing, (1997), 26(5)*

[8] Model/View Programming. *Qt Documentation* Consultado en https://doc.qt.io/qt-5/model-view-programming.html

[9] Johnson, Tomi H; Clark, Stephen R; Jaksch, Dieter. What is a quantum simulator? EPJ Quantum Technology, (2014), 1(1)

[10] GitHub - qosf/awesome-quantum-software: Curated list of open-source quantum software projects. Consultado en https://github.com/qosf/awesome-quantum-software

[11] Qiskit Overview. Consultado en https://qiskit.org/overview/

[12] Cirq. Consultado en https://quantumai.google/cirq

[13] Arute, F., Arya, K., Babbush, R. et al. Quantum supremacy using a programmable superconducting processor. Nature 574, 505510 (2019).

[14] Tabla de precios por uso sobre el servicio Braket de Amazon Web Services. Consultado en https://aws.amazon.com/es/braket/pricing/

[15] Entrevista al matemático Gil Kalai, publicado el 7 de febrero de 2018. Consultado en https://www.quantamagazine.org/gil-kalais-argument-against-quantum-computers-20180207/

[16] Martonosi, M. ; Roetteler, M. Next Steps in Quantum Computing: Computer Science's Role. A Computing Community Consortium (CCC) workshop report, 2019

[17] Vista general del servicio Braket de Amazon Web Services. Consultado en https://aws.amazon.com/es/braket/

[18] Marquezino, F., Portugal, R., Lavor, C., & de Lima Marquezino, F. *A Primer on Quantum Computing* Springer Publishing. (2019)

[19] Descripción detallada del elemento QWidget para el framework utilizado PyQt. Consultado en https://doc.qt.io/qt-5/qwidget.html#details

[20] Robinson, K. Here's how quantum computing could transform the future. Consultado en https://www.businessinsider.com/quantum-computing-investing-computers-enterprise-2021-3

[21] Palacios-Berraquero C, Mueck L, Persaud D Instead of supremacy use quantum advantage Nature, (2019), 576(7786)

[22] Página oficial de Python Consultado en https://www.python.org/

[23] Página oficial de PyQt Consultado en https://riverbankcomputing.com/software/pyqt/intro

[24] Página oficial de Visual Studio Code Consultado en https://code.visualstudio.com/

[25] Página oficial de Github Consultado en https://github.com/

[26] Página oficial de Notion Consultado en https://www.notion.so/product

[27] Página oficial de Mockflow Consultado en https://www.mockflow.com/

[28] Página oficial de GenMyModel Consultado en https://www.genmymodel.com/

[29] Página oficial de Microsoft Project Consultado en https://www.microsoft.com/es-es/microsoft-365/project/project-management-software

[30] Vista general del servicio Braket de Amazon Web Services Consultado en https://aws.amazon.com/es/braket/

[31] Formulario para solicitar acceso al uso de hardware cuántico de Google Quantum AI Consultado en https://docs.google.com/forms/d/1DfUWu4zUAJ87GKy-ZoTHrFri5IwIteKtMxKfsy3lmHE/viewform?edit_requested=true

[32] Guía de diseño y programación de aplicaciones de Windows Consultado en https://docs.microsoft.com/es-es/windows/apps/design/

[33] Lean Canvas, un lienzo de modelos de negocio para startups Consultado en https://javiermegias.com/blog/2012/10/lean-canvas-lienzo-de-modelos-de-negocio-para-startups-emprendedores/

[34] Yanofsky, Noson S. and Mannucci, Mirco A. *Quantum Computing for Computer Scientists* Cambridge University Press. (2018)

[35] Conceptos clave de la computación cuántica. Consultado en https://www.technologyreview.es/s/10920/que-es-un-ordenador-cuantico-definicion-y-conceptos-clave

[36] Página de la librería Pillow Consultado en https://python-pillow.org/

[37] Documentación de Qiskit Terra API Consultado en https://qiskit.org/documentation/apidoc/terra.html

[38] Documentación de Qiskit IBM Quantum Provider API Consultado en https://qiskit.org/documentation/apidoc/ibmq-provider.html

Glosario

cúbit El bit cuántico, o cúbit, es la unidad mínima de información para la computación cuántica. Se hará referencia indistintamente a cúbit o qubit, sin embargo, Fundéu indica que la grafía recomendada es cúbit. 4, 12

decoherencia cuántica Proceso por el cual un estado cuántico entrelazado pierde la pureza de su estado de manera que muestra comportamientos clásicos. 4, 13

entrelazamiento cuántico Propiedad física que ocurre cuando los dos miembros de un par de partículas (o cúbits) existen en un mismo estado cuántico. De esta manera, el cambio de estado de una partícula afectará de manera inmediata a la otra sin importar la distancia que las separe. 5, 12

superposición Propiedad de los cúbits que permite estar simultáneamente en múltiples estados, es decir, combinaciones de 1 y 0 a la vez. 5, 12

supremacía cuántica Es el punto donde el cálculo matemático realizado por un computador cuántico es, de manera demostrable, inalcanzable por el supercomputador clásico más potente. Debido a las connotaciones negativas que ha tenido el término 'supremacía' históricamente, se recomienda el uso del término ventaja cuántica [21]. 1

ventaja cuántica Es el punto donde el cálculo matemático realizado por un computador cuántico es, de manera demostrable, inalcanzable por el supercomputador clásico más potente. 1, 4, 8

volumen cuántico Métrica que mide el rendimiento y los índices de error de un ordenador cuántico. Pretende representar el rendimiento de manera universal, ya que este tipo de ordenadores son difíciles de comparar. Cuanto más grande, problemas más complejos puede resolver el ordenador . 8

Siglas

API Interfaz de programación de aplicaciones. 9, 40

CMD Símbolo del sistema. 39

GUI Interfaz gráfica de usuario. 39

HIP Horas Ideales de Programación. 33, 34

MoSCoW *Must Should Could Won't*. 33

MVC Modelo-Vista-Controlador. 22

MVP Producto Mínimo Viable. 3, 35, 36, 37

NISQ Noisy Intermediate-Scale Quantum. 10

PA Pruebas de Aceptación. 31, 32

PMBOK Guía de los fundamentos para la dirección de proyectos. 28

RSA Rivest, Shamir y Adleman, creadores del sistema criptográfico de clave pública más utilizado actualmente. La seguridad radica en el problema de la factorización de números enteros. 1

SDK Kit de desarrollo software (Software Development Kit). 6, 40

UT Unidad de Trabajo. 31, 32, 33, 36

APÉNDICE A
Manual de la aplicación desarrollada

A.1 Ventana principal

En este anexo se incluyen las capturas de las principales pantallas de la aplicación del simulador y una breve descripción de ellas.

En la ventana principal se puede observar la parte donde está el circuito que ocupa la mayor parte de la zona central. A su izquierda hay una caja blanca con el conjunto de puertas disponibles para arrastrar y soltar en la zona del circuito, concretamente sobre una línea de los cúbits. Si se pulsa sobre una de las imágenes de las puertas cuando están en la caja se abre una ventana con la información de la puerta A.3. En esta misma caja, si se pulsa sobre la segunda pestaña aparecerán las puertas creadas por el usuario mediante el botón ubicado en la parte superior derecha Crear puerta. Este botón redirige a la figura A.1. Justo a la izquierda se encuentra el botón para añadir un cúbit que al pulsarlo, crea una nueva línea en el circuito que representará el nuevo cúbit. Finalmente, el botón Simular, que al pulsarlo se recogerá la información del circuito en pantalla, se harán los cálculos correspondientes y se mostrarán los resultados A.4.

En la barra de menú de la parte superior tenemos las siguientes opciones:

- Archivo
  - Nuevo, crea un nuevo circuito en blanco.
  - Cargar, carga un fichero desde el explorador de archivos que contendrá el circuito guardado.
  - Guardar, guarda la información de un circuito en un fichero.
  - Salir, cierra la aplicación.
- Ejecutar
  - IBM, inicia el proceso de ejecutar el circuito en un ordenador cuántico, en este caso del proveedor IBM. Cuando finalice la ejecución se abre la ventana de resultados.

A.2 Ventana de creación de puerta lógica

En la siguiente imagen se muestra un formulario con el que se puede crear una puerta cuántica personalizada. Los datos a introducir son un nombre, una abreviatura en

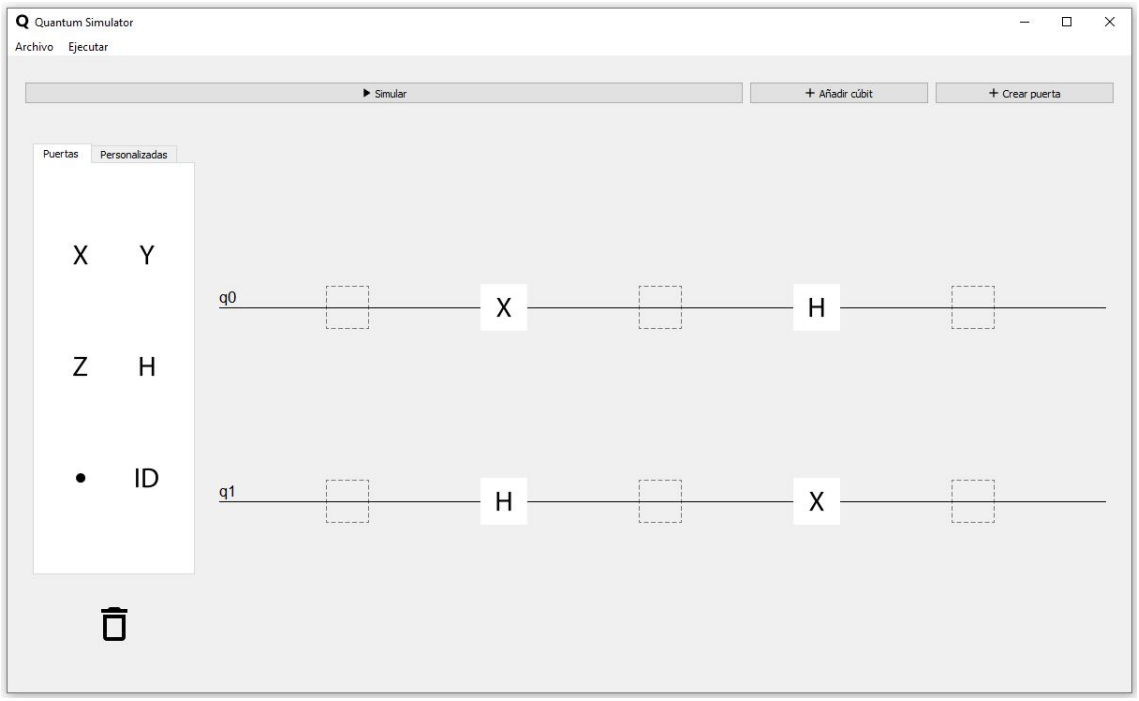


Figura A.1: Ventana principal de la aplicación

mayúsculas del nombre o letras con las que se creará la imagen que la identifique y la matriz correspondiente. Una vez introducidos los datos, al pulsar en Aceptar se añadirá a la pestaña de Personalizadas de la caja de puertas en la ventana principal.

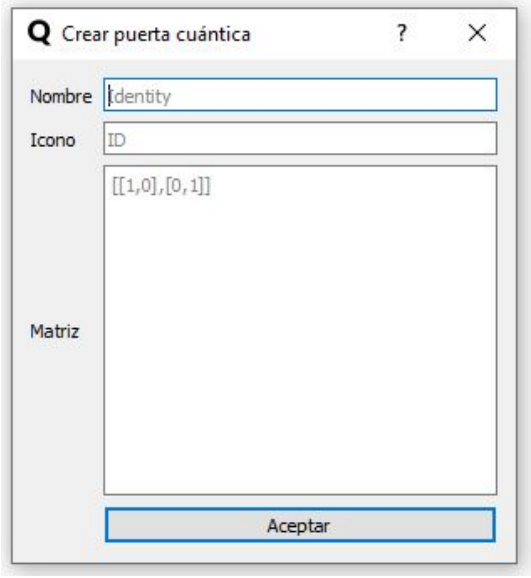


Figura A.2: Ventana de creación de una puerta lógica

A.3 Ventana de la información de puerta lógica

En esta ventana se puede consultar la información de las puertas lógicas. Se puede consultar su nombre, su icono y su matriz asociada.

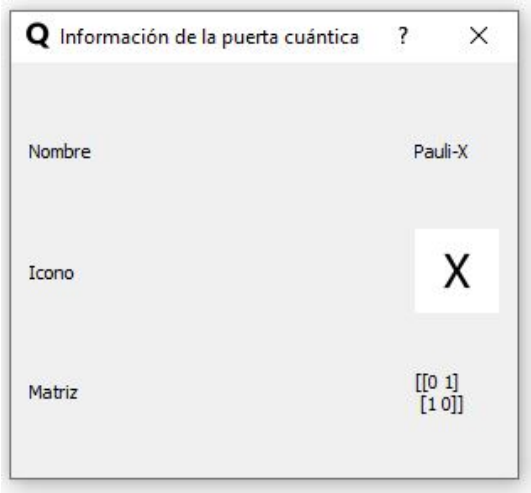


Figura A.3: Ventana de información de una puerta lógica

A.4 Ventana de resultados

En la pantalla de resultados, se puede ver la información de la simulación o ejecución realizada del circuito actual. Aquí se observan los posibles estados y la probabilidad de aparición de cada uno de ellos.

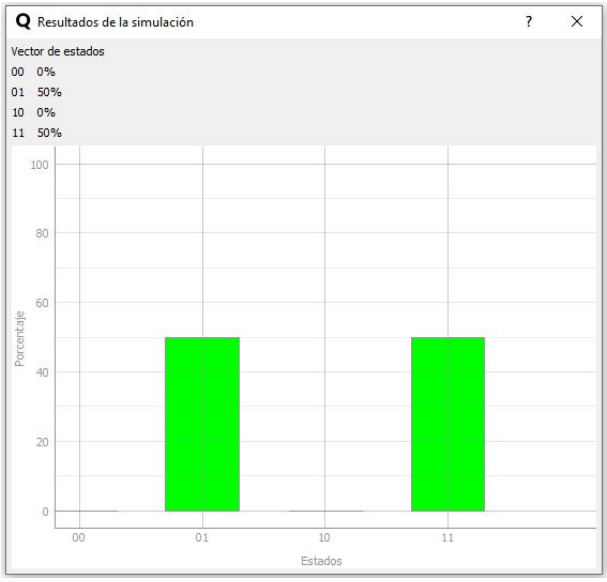


Figura A.4: Pantalla de resultados de la simulación