Matematikai és Informatikai Intézet

Tanulás segítő program és applikáció egy kvantumbites logikai kapukhoz

Készítette
Bakos Rózsa Ajándék
Programtervező informatikus BSc

Témavezető
Biró Csaba
Egyetemi docens

Eger, 2024

Tartalomjegyzék

Bevezetés

A kvantuminformatika térhódítása egyre nagyobb figyelemnek örvend, valamint új lehetőségei hatalmas potenciállal bírnak a számítástechnika terén. A kvantummechanika alapelveinek felhasználása új típusú megoldásokat eredményez, amelyek képesek áthidalni a jelenleg is használt számítógépek korlátait. Ennek köszönhetően ezek a kvanumszámítógépek olyan problémák megoldásában ígérkeznek hatékonyabbnak, amelyek a hagyományos számítógépek számára nehezen, vagy egyáltalán nem megoldhatók.

A kvantumszámítógépek potenciális alkalmazási területei közé tartozik a mesterséges intelligencia, kriptográfia, gyógyszerkutatás és a számításelmélet. Azonban az ilyen rendszerek működése rendkívül érzékeny a környezeti tényezőkre, például gyakran használnak extrém alacsony hőmérsékletet követelő szupravezetőket. A jelenleg létező kvantumszámítógépek egyelőre kezdeti fázisban járnak, de a különböző cégek, kutatócsoportok között kialakult verseny ezen a helyzeten bármikor változtathat.

A kvantum-számítástechnika egyik kulcsfontosságú területe a kvantum logikai kapuk koncepciója, amelyek a kvantum bitek (más néven qubit) manipulációját teszik lehetővé. Ehhez a már említett kvantummechanika alapelveire támaszkodnak. Értelmezésük, valamint a hozzá tartozó összetett matematikai háttér sok esetben nehézséget okozhat.

Szakdolgozatom célja, hogy bemutasson ezen problémák áthidalására egy olyan tanulás segítő programot, mely közelebb hozza az érdeklődőkhöz a kvantumkapuk koncepcióját. Ezt egyqubites kapuk bemutatásával teszem meg. Az alkalmazás tervezése és implementálása során figyelembe veszem a felhasználók igényeit és a pedagógiai célokat, miközben kihasználom a kvantumtechnológia által nyújtott lehetőségeket.

A továbbiakban bemutatom a kvantuminformatika alapjait, a kvantumlogikai kapuk működését és jellemzőit, valamint részletesen ismertetem a fejlesztett tanulás segítő programot, beleértve annak tervezési alapelveit, implementációját és tesztelését. Végül összefoglalom az elért eredményeket és felvázolom a jövőbeli kutatási irányokat ezen a területen.

1. fejezet

Kvantuminformatika

1.1. Előzmények

A kvantummechanika gyökerei az 1800-as évek elejére vezethetők vissza, ahol a fizika többek között olyan problémákkal is küzdött, mint a hőmérsékleti sugárzás.

Az első matematikai modellt, amely erre magyarázatot adott, Max Planck német fizikushoz kötjük. A ma már Planck-féle kvantumhipotézisnek ismert levezetés 1900-ban történt előterjesztésre. Ezt a dátumot tekintjük a "régi" kvantumelméleti korszak kezdetének, valamint a kvantumfizika születésének. A jelenséggel Albert Einstein is foglalkozott, aki 1905-ben megjelent, fényelektromos jelenségről szóló cikkében a már említett hőmérsékleti sugárzástól független is alkalmazta a Planck-féle kvantumhipotézist.

Ezután rohamos növekedésnek kezdett a tudományág. A "régi" kvantumelméleti korszak végét 1924-re datáljuk, amikor De Brogile publikálta anyaghullámokról szóló elméletét. A modern kvantummechanika születése 1925-ben történt, jelenleg is ebben a korszakban vagyunk.

Természetesen az informatikát sem kerülte el ez a hullám. 1981-ben Richard Feynman előadásában felvetette egy kvantumelveken működő számítógép ötletét, ami képes kvantumrendszereket szimulálni. Az ő nevéhez fűződik a kvantumszámítógép kifejezés. A korszak hasonlóan fontos neve még Paul Benioff, aki 1982-ben publikált, tanulmányában lefektette a kvantumszámítási modellek alapjait. Feynman és Benioff gondolatait felhasználva 1985-ben David Deutsch bemutatta tanulmányában egy kvantumszámítógép elméleti működését, ami képes elvégezni a klasszikus számítógép számításait, de kihasználva a kvantummechanika előnyeit. Fontos személy volt még ebben az időszakban Peter Shor, aki 1994-ben megalkotott egy olyan kvantumalgoritmust, ami a napjainkban használt titkosítási eljárásokat veszélybe sodorhatja. Az 1994-2000-es időszakban számos hasonlóan fontos kvantumalgoritmusok született. Ilyen például az 1996-ban készült Grover algoritmus, ami a rendezetlen adatokban való keresést segítette.

Ahogy ezek a felfedezések egyre tovább bővítették a kvantumszámítás fogalmát, úgy felmerült az igény a kvantumlogikai kapukra is. Az 1980-as évek végétől kezdve számos kaput állítottak elő.

1.2. Jelenlegi helyzet

A 21. században kialakult egy verseny a kutatóintézetek és tech óriások közt, cél egy kvantumszámítógép megépítése volt. Ehhez a korszakhoz fűződik a kvantumfölény kifejezés is. Ez arra utal, hogy egy kvantumszámítógép képes megoldani egy olyan problémát, melyre klasszikus társai nem képesek lehetséges időn belül.

2011-ben a D-Wave Systems cég volt az első, akik azt állították, hogy megépítették az első kereskedelmi forgalomban kapható kvantumszámítógépet, a D-Wave One-t, ami 128 qubites lapkakészleten működött.

Fontos esemény volt még a 2019-ben a Google által bejelentett kvantumfölény. 53 qubites Sycamore processzorával rendelkező kvantumszámítógépük 200 másodperc alatt megoldott egy konkrét problémát, amelyhez egy klasszikus szuperszámítógépnek körülbelül 10 000 év alatt végzett volna el. Ugyanebben az évben jelent meg az IBM első kvantumszámítógépe is, az IMB Quantum System One.

Ebben az időszakban elért jelentős előrelépés ellenére számos kihívás még mindig fennáll, ilyen például a kvantumbitek minősége és stabilitása. Az olyan cégek, mint az IBM, Google és a Microsoft továbbra is jelentős összegeket fektet a kvantumszámítógépek fejlesztésébe. Ezen kívül számos ország és nagyhatalom is foglalkozik ezzel a tudományággal.

Hazai viszonylatban is folyamatos kutatások történnek, ezeket különböző egyetemek mellett a Kvantuminformatikai Nemzeti Laboratórium és a HunQuTech konzorcium vezeti. Céljuk elérni a nemzetközi szintet.

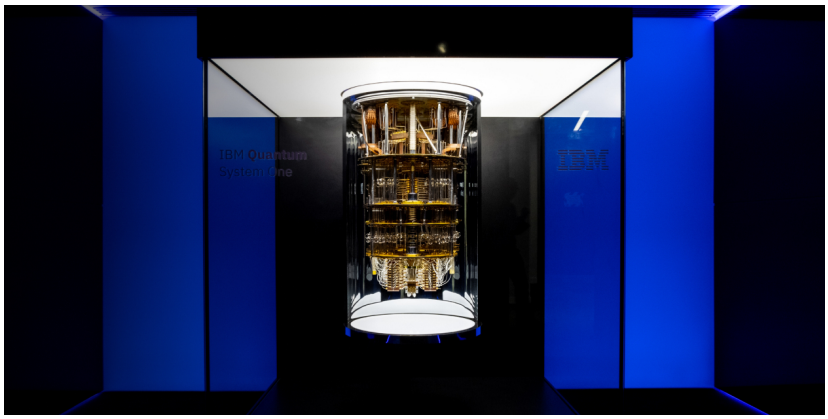
Ebben az időszakban több megközelítés is született a kvantumszámítógépekre, ezáltal több típusról is beszélhetünk, például:

1. Ioncsapdás: A kvantumbitek a csapdában lévő részecske állapotai alapján definiálódnak.

2. Szupravezetős: Szupravezető áramköröket használnak a qubitek létrehozására és manipulálására. Rendkívül alacsony hőmérsékleten, az abszolút nulla közelében működnek, hogy fenntartsák állapotukat. Például az IBM Q System One is ilyen.

3. Fotonikus: Fotonokat használnak kvantumbitként. Nagy sebességű kommunikációra és információfeldolgozásra képesek.

4. Quantum Annealing: Speciális kvantumszámítógép, Optimalizálási problémák megoldására tervezték.



1.1. ábra. IBM Q System One

1.3. Nehézségek

Mint láttuk, a kvantumszámítógépek jelenleg is kísérletezés alatt állnak, valamint számos kihívások továbbra is feltáratlanok maradtak. Egy kvantumszámítógép építése és karbantartása, azok speciális igényük és építőelemeik miatt mai napig nagy költségeket ölelnek fel.

A kvantumbitek minősége és stabilitása szintén egy jelentős probléma. A jelenlegi kvantumszámítógépek kevés qubittel rendelkeznek, skálázhatóságuk limitált, emiatt szűk körben használhatóak, számos komplex feladat megoldására továbbra is képtelenek.

Környezetre való érzékenysége miatt, extrém környezetekben tárolják őket. Az olyan tényezők, mint például a zaj, számítási problémákat okozhat, amik kijavítása rendkívül időigényes.

1.4. Előnyök és veszélyek

Nehézségei ellenére a kvantumszámítógépek számos lehetőséget is hordoznak magukban. Sokszor viszont ezek az előnyök rossz kezekben képesek veszélyeket is okozni.

Legfontosabb potenciáljuk a gyorsaságuk. Egy kvantumszámítógép sokkal gyorsabban képes elvégezni számításokat, mint a klasszikus számítógépek. Ez annak köszönhető, hogy egy klasszikus számítógép szekvenciálisan végzi feladatait, ezzel szemben a kvantumszámítógépek párhuzamosságra képesek.

Bár számos komplex feladat megoldására jelenleg is képtelenek, léteznek olyan összetettebb problémák, amiben a kvantumszámítógépek megfelelőnek bizonyultak, míg a klasszikus számítógépek számára megoldhatatlanok voltak. Ilyenek például különböző optimalizálási problémák, illetve faktoriális számítás. Ezek mögött számos kvantumalgoritmus jelenik meg, amely hasonlóan egy új ága a kvantuminformatikának.

A gyorsasági és számítási előnyei miatt több más iparág segítésére is szolgálhatnak a kvantumszámítógépek, többek között az anyag- és a gyógyszerkutatás is ide tartozik.

Viszont ezek az előnyök veszélyeket is hordozhatnak magukban. A már említett Shor algoritmus egy szám prímtényezős felbontását végzi el hatékonyan. A problémát az okozza, hogy a jelenlegi titkosítási algoritmusok (például az RSA) nagy része emiatt védtelen lesz, mivel ezek prímszámokat használnak titkosításra. Természetesen ezekre a problémákra is léteznek kutatások, így született a kvantumkriptográfia területe, ami fejlettebb biztonságot ígér.

A sok új lehetőség és a felsorolt előnyök nem léteznének a kvantum logikai kapuk nélkül, amik az alapkövei a kvantumszámítógépeknek.

2. fejezet

Klasszikus- és kvantum logikai kapuk

2.1. Klasszikus logikai kapuk és áramkörök

Elektromos impulzusokhoz értékeket társítunk, az alapján hogy küldtünk-e vagy sem. Ha érzékelünk, akkor ezt I logikai értéknek vagy 1 bitnek tekintjük. Ellenkező esetben H logikai értéknek, vagy 0 bitnek feleltetjük meg.

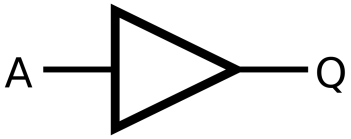
Ezekhez az impulzusokhoz többnyire logikai kapukat társítunk, melyek bináris operátorokat foglalnak magukban. A logikai kapuk alapvető építőkövei az elektronikának és számos célra használják őket. Összekapcsolásukkal áramköröket alakíthatunk ki, amik lineárisak és balról jobbra értelmezzük őket. A bal oldali vezetékek jelentik a bemenetet, míg a jobb oldaliak a kimenetet. Az ismertebb kapukhoz speciális ábrák és igazságtáblák tartoznak.

Az igazságtáblák a klasszikus logika alapvető eszközei, amelyek segítségével értelmezhetjük az adott műveleteket, valamint ellenőrizhetjük áramköreinket. Megmutatják az összes lehetséges bemeneti kombinációt, illetve a műveletek alkalmazása után a várható kimenetet is. Ezek a logikai műveleteket, kifejezéseket Boole-algebrának nevezzük, amely egy 19. századi matematikus, George Boole nevét viseli

2.1.1. Logikai kapuk

Buffer

A buffer kapuk kimenete megegyezik a kimenetükkel, egy biten értelmezzük. Jele: A



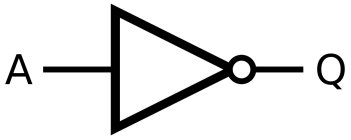
2.1. ábra. Buffer kapu rajza

A	A
0	0
1	1

2.1. táblázat. Buffer kapu igazságtáblája

NOT, vagy Negáció

A negáció kapu (vagy NOT) hasonlóan egy bites, a bemeneti jel logikai értékét megfordítja a kimeneten. Jele:$\neg A$



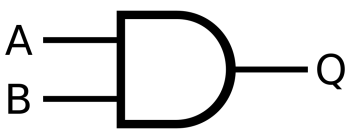
2.2. ábra. NOT kapu rajza

A	$\neg A$
0	1
1	0

2.2. táblázat. Negáció kapu igazságtáblája

AND, vagy Konjukció

Más néven logikai és. Kétbites művelet, kimenete akkor igaz, ha mind a két operandusa igaz. Jele: $A \wedge B$



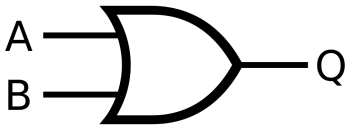
2.3. ábra. AND kapu rajza

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

2.3. táblázat. A konjukció igazságtáblája

OR, vagy Diszjunkció

Más néven logikai vagy. Szintén kétbites művelet, értéke csak akkor hamis, ha mind a két operandusa hamis. Jele: $A \vee B$



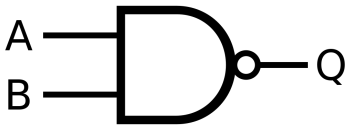
2.4. ábra. OR kapu rajza

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

2.4. táblázat. A diszjunkció igazságtáblája

NAND, vagy Negált konjukció

A konjukció negált változata, értéke akkor hamis, ha mindkét operandusa igaz. Jele: $\neg(A \wedge B)$



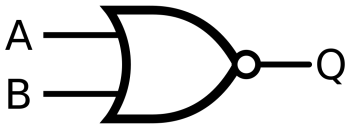
2.5. ábra. NAND kapu rajza

A	B	$\neg(A \wedge B)$
0	0	1
0	1	1
1	0	1
1	1	0

2.5. táblázat. A negált konjukció igazságtáblája

NOR, vagy Negált diszjunkció

A diszjunkció negált változata, értéke akkor igaz, ha mindkét operandusa hamis. Jele: $\neg(A \vee B)$



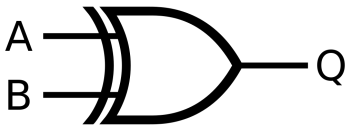
2.6. ábra. NOR kapu rajza

A	B	$\neg(A \vee B)$
0	0	1
0	1	0
1	0	0
1	1	0

2.6. táblázat. A negált diszjunkció igazságtáblája

XOR, vagy Exclusive OR

Más néven kizáró vagy. Értéke akkor hamis, ha a bemenetek megegyeznek. Jele: $A \oplus B$



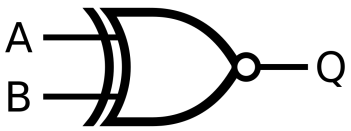
2.7. ábra. XOR kapu rajza

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

2.7. táblázat. A negált konjukció igazságtáblája

XNOR, vagy Exclusive NOR

A kizáró vagy negáltja, értéke akkor hamis, ha a bemenetek különböznek Jele: $\neg(A \oplus B)$



2.8. ábra. XNOR kapu rajza

A	B	$\neg(A \oplus B)$
0	0	1
0	1	0
1	0	0
1	1	1

2.8. táblázat. A negált konjukció igazságtáblája

2.1.2. Univerzális kapuk

Léteznek olyan logikai kapuk, amelyek képesek reprodukálni bármely másik működését. Ezek az univerzális kapuk lehetővé teszik akármelyik logikai függvény leképezését, ami azt jelenti, hogy akár tetszőleges áramkör is megvalósítható velük.

Bizonyítható, hogy bármilyen logikai függvény összeállítható csak NOT és AND, vagy NOT és OR függvények kombinációival. Mint láttuk, ezek a kapuk ötvözhetőek, erre szolgálnak a NAND és NOR kapuk.

Ebből következtethető, hogy bármilyen Boole-függvény megvalósítható olyan áramkörrel, amely csak NAND vagy NOR kapukat alkalmaz. Gyakran emiatt előnyt élveznek, mivel más kapuknak nincs ilyen tulajdonsága.

2.2. Kvantum logikai kapuk és áramkörök

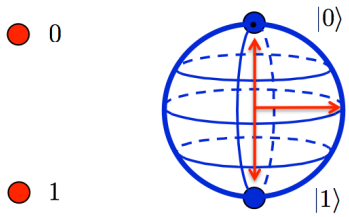
A kvantumkapuk jelentőségét a kvantummechanika alapelveire támaszkodva értelmezhetjük, amelyek olyan jelenségeket és viselkedéseket írnak le, melyek a klasszikus fizika keretein belül nem megfigyelhetők vagy magyarázhatók. A kvantummechanika alapján működő kapuk és áramkörök lehetővé teszik olyan számítások végrehajtását, amelyek rendkívül nagy számítási kapacitással és párhuzamosítással rendelkeznek, ami jelentős előnyöket kínál a hagyományos, klasszikus számítógépekkel szemben.

Bár vannak hasonlóságok a klasszikus változatokhoz képest, a kvantum logikai kapuk egy sokkal komplexebb matematikai hátteret és fizikai jelenségeket hordoznak magukban.

Jelen alfejezet csak pár, ismertebb logikai kaput tartalmaz, de fontos megemlíteni, hogy a kvantumkapuk száma végtelen. Mivel szakdolgozatom az egyqubites kapukra irányul, ezért a több kvantumbites változatok csak megemlítő jelleggel szerepelnek.

2.2.1. Qubit

A kvantumbit, röviden qubit, a kvantuminformatika alapvető építőeleme, és a klasszikus számítógépek működésétől eltérő, kvantummechanikai alapokon nyugvó adatábrázolási egység. Míg a hagyományos bit csak két állapotban (0 vagy 1) lehet jelen, a qubit képes egyszerre számos állapotban lenni, ami egyike annak a tulajdonságának, amely a kvantuminformatikát annyira különlegessé teszi.



2.9. ábra. A bit és qubit reprezentációja

Ahogy az a 2.9. ábrán látható, a qubit számos állapotát egy úgynevezett Bloch gömbön ábrázoljuk, északi pólusán $|0\rangle$, déli pólusán pedig $|1\rangle$ helyezkedik el. A $|\rangle$ és $\langle|$ jelöléseket braket-nek nevezzük, de Dirac jelölésként is ismert, és a kvantumállapotok jelölésében segít. A ket ($|\Psi\rangle$) egy oszlopvektort, a bra ($\langle\Psi|$) pedig egy sorvektort jelöl.

A kvantummechanika alapelvei és a qubitok sajátos tulajdonságai lehetővé teszik olyan számítási feladatok elvégzését, amelyek a hagyományos számítógépek számára gyakorlatilag lehetetlenek lennének.

2.2.2. Szuperpozíció és összefonódás

Egy qubit állapotát a következőképpen adhatjuk meg:

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{2.1}$$

ahol α és β komplex számok, valamint teljesül, hogy $|\alpha|^2+|\beta|^2 = 1$. Ez azt jelenti, hogy a $|\Psi\rangle$ kvantumbit $|\alpha|^2$ eséllyel 0, $|\beta|^2$-el pedig 1 lesz. Ezt szuperpozíciónak nevezzük. A két szélső érték, azaz $|0\rangle$ vagy $|1\rangle$ akkor áll elő, ha α 0 vagy 1, β pedig ennek ellentettje. A szuperpozíció legjobban a Schrödinger macskája gondolatkísérlettel prezentálható, melyet Erwin Schrödinger fogalmazott meg 1935-ben.

A qubit másik fontos jelensége a kvantum-összefonódás. Ez egy olyan jelenség a kvantummechanikában, amelyben két vagy több részecske állapota olyan összekapcsolt módon van, hogy az egyiken végzett mérések azonnal hatnak a másikra, függetlenül attól, hogy a részecskék milyen fizikai távolságra vannak egymástól.

2.2.3. Qubit mérése

Méréskor eredményként egyetlen klasszikus bitet kapunk. A qubit mérése során az eredmény kvantumállapota "összeomlik" az adott értékre, amelyet a mérés során megfigyeltünk. Azaz elveszti a szuperpozícióját és az összefonódottságát, ami azt jelenti, hogy a mérés után a rendszer elveszíti a kvantumos jellegét, és klasszikus állapotba kerül. Ez az összeomlás a kvantummechanika alapvető jelensége, amely lehetővé teszi a kvantumrendszer állapotának rögzítését a mérés pillanatában. Emiatt a mérések kritikus szerepet játszanak.

2.10. ábra. Mérés rajza

2.2.4. Kvantum logikai kapuk

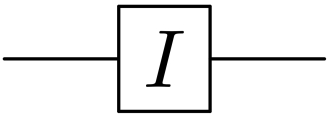
I, vagy Identity

A buffer kapuhoz hasonlóan nem végez műveletet, helyben hagyja a kvantumbiteket. Egy qubiten használjuk. Azonosság transzformációként működik, alakja:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.2}$$

Ez egy tetszőleges kvantumbiten:

$$|\Psi\rangle = I\,|\varphi\rangle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = a\,|0\rangle + b\,|1\rangle \tag{2.3}$$



2.11. ábra. I kapu rajza

X, vagy Pauli-X

A klasszikus NOT kapunak felel meg, bit-flipnek is hívják. A Bloch gömb X tengelyére tükröz. Egy qubiten működik. Alakja:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.4}$$

Ez egy tetszőleges kvantumbiten:

$$|\Psi\rangle = X\,|\varphi\rangle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = b\,|0\rangle + a\,|1\rangle \tag{2.5}$$

Az X kapu hatására a valószínűségi amplitúdók cserélődnek. Inverze saját maga, tehát például:

$$|\Psi\rangle = XX\,|0\rangle = |0\rangle \tag{2.6}$$

2.12. ábra. X kapu rajza

Y, vagy Pauli-Y

Az X kapuhoz hasonlóan felcseréli a $|0\rangle$-t és az $|1\rangle$-t, viszont az Y kapu a relatív fázist is megváltoztatja. A Bloch gömb Y tengelyére tükröz. Egy qubiten működik. Alakja:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{2.7}$$

Ez egy tetszőleges kvantumbiten:

$$|\Psi\rangle = Y\,|\varphi\rangle \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = -ib\,|0\rangle + ia\,|1\rangle\,, \tag{2.8}$$

ahol i komplex szám. Inverze saját maga, tehát például:

$$|\Psi\rangle = YY\,|0\rangle = |0\rangle \tag{2.9}$$

2.13. ábra. Y kapu rajza

Z, vagy Pauli-Z

Phase-flip kapunak is hívják, ebből adódóan a fázist cseréli meg. A Bloch gömb Z tengelyére tükröz. Egy qubiten működik. Alakja:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.10}$$

Ez egy tetszőleges kvantumbiten:

$$|\Psi\rangle = Z\,|\varphi\rangle \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = a\,|0\rangle - b\,|1\rangle \tag{2.11}$$

Látható, hogy a Z kapu csak az $|1\rangle$ valószínűségi amplitúdót változtatja. Inverze saját maga, tehát például:

$$|\Psi\rangle = ZZ\,|0\rangle = |0\rangle \tag{2.12}$$

2.14. ábra. Z kapu rajza

H, vagy Hadamard

A Hadamard kapu segítségével a már említett szuperpozíciós állapotokat lehet előállítani. Egy qubiten működik. Alakja:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.13}$$

Ez egy tetszőleges kvantumbiten:

$$|\Psi\rangle = H\,|\varphi\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \frac{a+b}{\sqrt{2}}\,|0\rangle + \frac{a-b}{\sqrt{2}}\,|1\rangle \tag{2.14}$$

Mivel a kaput gyakran használják a standard bázisvektorokon, emiatt külön számon tartják őket:

$$|\Psi\rangle = H\,|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.15}$$

$$|\Psi\rangle = H\,|1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.16}$$

A Hadamard kapu inverze saját maga, tehát például:

$$|\Psi\rangle = HH\,|0\rangle = |0\rangle \tag{2.17}$$

2.15. ábra. H kapu rajza

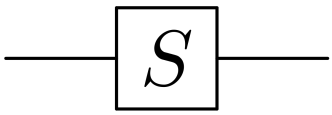
S, vagy Phase

Más néven fáziseltolási kapu. $\pi/2$ fáziseltolást alkalmaz a qubit állapotára. A Bloch gömb körül az óramutató járásával megegyező irányban forgat $\pi/2$ radiánnal. Egy qubiten működik. Alakja:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \tag{2.18}$$

Ez egy tetszőleges kvantumbiten:

$$|\Psi\rangle = S\,|\varphi\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = a\,|0\rangle - ib\,|1\rangle \tag{2.19}$$



2.16. ábra. S kapu rajza

T, vagy $\pi/8$

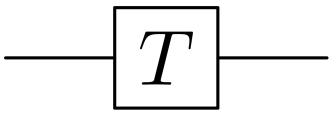
A T kapu, hasonlóan az S kapuhoz, fáziseltolást alkalmaz egy qubit állapotára. Az állapotvektort $\pi/4$ radiánnal forgat a Bloch gömb körül, óramutató járással megegyező irányban. Alakja:

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{2.20}$$

Ez egy tetszőleges kvantumbiten:

$$|\Psi\rangle = T\,|\varphi\rangle = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = a\,|0\rangle + e^{i\pi/4} b\,|1\rangle \tag{2.21}$$

Az S kapuval való azonosságok miatt elmondható, hogy $S = T^2$

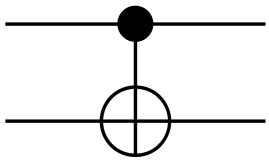


2.17. ábra. T kapu rajza

CNOT, vagy Controlled Not

A CNOT kapu két qubites művelet, ahol az első qubit vezérlő-, a második pedig cél kvantumbit néven ismert. Ha a vezérlő $|1\rangle$, akkor a cél qubiten X kaput használ, a többi változatlan. Amennyiben a vezérlő qubit $|0\rangle$, a cél változatlan marad. Klasszikus logikai kapuként is alkalmazható. Inverze saját maga. Alakja:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.22}$$

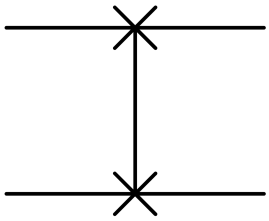


2.18. ábra. CNOT kapu rajza

SWAP

Két qubites kvantumkapu, mely felcseréli a bemenetek állapotát. Mivel három CNOT kapuval valósítható meg, így más jelölést is alkalmaznak rá, az egyszerűbb változat a 2.19. ábrán látható. Inverze saját maga. Alakja:

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

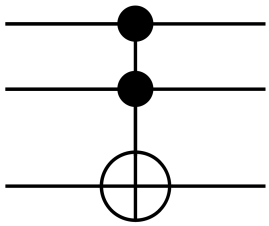


2.19. ábra. SWAP kapu rajza

Toffoli

Ahogy az a 2.20. ábrán látható, a Toffoli kapu CNOT műveletet használ, ezért szokás CCNOT kapunak is nevezni. Három qubiten működik, az első kettő vezérlő, az utolsó pedig a cél qubit. Klasszikus logikai kapuként is alkalmazható, ez esetben univerzális klasszikus kapunak tekintik. Inverze saját maga. Alakja:

$$Toffoli = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.24}$$



2.20. ábra. Toffoli kapu rajza

2.2.5. Univerzális kvantumkapuk

Klasszikus esetben tapasztaltuk, hogy léteznek univerzális kapuk. Ez amiatt lehetséges, mivel csak végesen sok Boole-függvény van egy adott számú változó esetén. Kvantumkapuknál elmondható, hogy a lehetséges kapuk száma végtelen sok. Emiatt a fontos különbség miatt univerzális kvantumkapuk halmaza sem létezik. Ellenben, ha a kapuknak vesszük egy véges számú gyűjteményét, akkor beszélhetünk univerzális megoldásokról, viszont ezek sem használhatóak minden lehetséges kvantumáramkörre, csak hozzávetőlegesen.

2.3. Reverzibilis kapuk

Megfordítható kapuknak is nevezik őket. Két alapvető tulajdonsága van:

1. Az eredményükből egyértelműen kikövetkeztethetőek a bemenetek, azaz egy bemeneti kombinációhoz pontosan egy kimeneti kombináció tartozik, és fordítva

2. Lehetővé teszik a bemenetek helyreállítását a kimenetekből anélkül, hogy információt veszítenének.

Vegyük például az OR kaput. Mint láttuk, értéke csak akkor hamis, ha bemenetei hamisak. Viszont minden más esetben nem tudjuk megállapítani, hogy melyik volt az aktuális bemenet a háromból, hacsak nincs további információnk. Tehát az OR kapuról elmondható, hogy irreverzibilis (nem reverzibilis) kapu. A legtöbb klasszikus logikai kapu szintén ebbe a csoportba tartozik, a már említett kapuk közül ez alól a CNOT és a Toffoli kivétel.

Lehetőség van klasszikus logikai művelet végrehajtására csak reverzibilis kapuk használatával, de ilyen esetekben gyakran szükségessé válik a segédbitek használata.

Alapvetően bármilyen reverzibilis klasszikus kapu megvalósítható kvantumszámítógépeken, így emiatt létezik a Toffoli és CNOT kapunak kvantum változata is.

A klasszikus logikai kapukkal ellentétben, a kvantum kapuk közül mindegyik reverzibilis.

3. fejezet

Tanulás segítő program és applikáció

3.1. Technológiai körültekintés

A különböző technológiák és eszközök közötti választás nem mindig egyértelmű. Ezért szükséges részletesen mérlegelni és összehasonlítani az elérhető lehetőségeket. A nyelv, keretrendszer vagy platform kiválasztása előtt érdemes megfontolni a különböző szempontokat, mint például a teljesítmény, a fejlesztési idő és a támogatottság. Szakdolgozatom írásának kezdetekor, a nyelv és keretrendszer választásnál figyelembe vettem az alábbi elképzeléseket:

1. Két részből álljon: egy telefonos applikációból és egy asztali alkalmazásból, mindkettő angol nyelven.

2. Az egy kvantumbites kapuk egy csoportját mutassa be, ezeket interaktív, tanulást segítő formában.

3. A telefonos applikáció képes legyen csatlakozni az asztali alkalmazásra, majd szenzor adatokat küldeni. Fontos volt továbbá, hogy rövid ismertetőket tudjon a felhasználó olvasni az adott kapukról.

4. A program egy letisztult felületet nyújtson, ahol a felhasználó válogathat az adott egy kvantumbites kapuk közül. Ezekről kapja meg azt a leírást, ami a telefonon is elérhető, valamint animációkat is tudjon nézni róluk.

5. A program középpontja egy Bloch gömb legyen, egy állapotvektorral. Ezt a felhasználó tudja állítani, valamint amennyiben csatlakoztatott telefont, ki is tudja próbálni az adott kapukat. A telefon forgatásával képes legyen a cél állapotba igazítani a vektort, ez egy minimális hibatűréssel történjen.

A program lehetőségeit szűkítette a tény, hogy minél pontosabb ábrákat, reprezentációkat kellett keresnem. A gömb forgatását vettem a legmeghatározóbb tényezőnek választásnál. Egy egyszerű 3D-s objektumot több nyelv és keretrendszer is támogat. Kipróbáltam, hogy ez Java Fx-ben valamint PythonQt-ban hogy jelenik meg. Viszont, mint ahogy azt a 2.9. ábrán is látható, a Bloch gömb tartalmaz egy vektort, aminek pozíciója változhat, ami miatt a klasszikus megjelenítés nem elegendő, hiszen nem az objektumot forgatjuk. Emiatt fontos volt áttekintenem, hogy milyen kvantumáramkörökkel foglalkozó könyvtárakkal szolgálnak a keretrendszerek.

A telefonos applikáció kapcsán fontos volt, hogy a választott operációs rendszer minél több felhasználót lefedjen, valamint közel pontos szenzor adatot legyen képes küldeni.

3.2. Választott technológiák

3.2.1. Asztali alkalmazás

Python

A Python egy magas szintű, interpretált programozási nyelv, amely egyszerűségéről és könnyen olvashatóságáról ismert. Manapság a legnépszerűbb nyelv.

Szintaxisát úgy tervezték, hogy intuitív és olvasható legyen, így kezdők és tapasztalt programozók számára egyaránt nagyszerű választás. Hangsúlyozza a kód olvashatóságát, és lehetővé teszi a fejlesztők számára, hogy más nyelvekhez képest kevesebb kódsorban oldjanak meg problémákat.

A Python emellett egy interpreteres nyelv, ami azt jelenti, hogy a Pythonban írt kód soronként hajtódik végre, nem pedig előzetesen gépi kódba fordítva. Ez megkönnyíti a fejlesztést és a hibakeresést, mivel a kód azonnal végrehajtható külön fordítási lépés nélkül.

Dinamikus típusrendszerének köszönhetően, változótípusokra a rendszer futás közben következtet. Ez nagyobb rugalmasságot és gyorsabb fejlesztést tesz lehetővé.

A Python támogatja az objektum-orientált programozás (OOP) elveit, lehetővé téve a fejlesztők számára, hogy osztályokat és objektumokat hozzanak létre az adatok és a viselkedések egységbe zárásához.

Egy átfogó könyvtárral is rendelkezik, amely modulokat és funkciókat biztosít különféle feladatok végrehajtásához, például hálózatkezelés, matematikai kifejezések stb.

Mindezek miatt a Python nyelv rendkívül sokoldalú. Számos alkalmazáshoz használják, beleértve a webfejlesztést, adatelemzést, gépi tanulást, mesterséges intelligenciát, tudományos számítástechnikát, automatizálást stb.

PySide modul

A PySide lehetővé teszi a fejlesztők számára, hogy Qt alapú alkalmazásokat írjanak Python használatával, hozzáférést biztosítva a Qt keretrendszer összes funkciójához és szolgáltatásához. Előnyei:

1. Cross-platform: Hasonlóan a Qt-hoz, a Pyside is cross-platform, tehát az ezzel fejlesztett alkalmazások számos operációs rendszeren futhatnak, beleértve a Windowst, macOS-t és a Linuxot, anélkül, hogy a kódon jelentős változtatásokat kellene végrehajtani.

2. GUI fejlesztés: A PySide leegyszerűsíti a grafikus felhasználói felületek (GUI) fejlesztését widgetek létrehozására, elrendezésekre és események kezelésére, a Qt hatékony eszközkészletének segítségével.

Az alkalmazás Pyside6-ot használ, ami Qt6-ot támogat

Qiskit könyvtár

A legjobban elterjedt nyílt forráskódú Python könyvtár ami kvantumáramkörökkel foglalkozik, az IBM fejleszti. Segítségével különböző áramköröket alakíthatunk ki, és szimulálhatunk. Lehetőség van az adott qubit állapotokat Bloch gömbön is reprezentálni. Hasonló könyvtárakhoz képest sokkal kvantumáramkör és kvantumszámítás központúbb, ami az asztali alkalmazás magját adja.

Matplotlib könyvtár

A Qiskit-ben kialakított áramköröket és Bloch gömböket a Matplotlib könyvtárral könnyen meg lehet jeleníteni. A könyvtár széles körben biztosít megjelenítést különböző diagramokhoz, 2D és 3D objektumokhoz, és még sok máshoz. A Matplotlib egy népszerű Python-könyvtár, amelyet statikus, interaktív és animált vizualizációk létrehozására használnak Pythonban. Széles körben használják különféle területeken.

A Matplotlib könyvtárral előállított objektumokat egyszerűen hozzá lehet adni a PySide projektekhez.

Numpy könyvtár

A NumPy egy alapvető csomag a Python segítségével végzett komplex számításokhoz. Támogatja a többdimenziós tömböket és mátrixokat, valamint matematikai függvények gyűjteményét biztosítja az ezeken a tömbökön való műveletekhez.

Ahogy azt a 2.2.4. fejezetben láttuk, a komplex mátrix számítások elengedhetetlenek a kvantumlogikai kapuk prezentációjában.

Socket könyvtár

Pythonban a socket könyvtár alacsony szintű hálózati interfészt biztosít, amely lehetővé teszi a hálózati socketek létrehozását és az azokkal való interakciót. Megkönnyíti mind a kliens, mind a szerver hálózati alkalmazások létrehozását. Támogatja a TCP és UDP protokollokat.

Ez könyvtár a telefon csatlakoztatása miatt szükséges.

3.2.2. Applikáció

Android Studio

Android-alkalmazások fejlesztésének hivatalos integrált fejlesztői környezete (IDE). A Google fejlesztette, és a JetBrains IntelliJ IDEA szoftverén alapul.

Felhasználóbarát felülettel rendelkezik, amely különböző paneleket tartalmaz a kód szerkesztéséhez, az elrendezések megtervezéséhez, a projektfájlok kezeléséhez, valamint a naplók és hibák megtekintéséhez.

A felületet egy XML Layout Editor is segíti. Ez egy vizuális szerkesztőt tartalmaz az Android-alkalmazások felületeinek megtervezéséhez, XML-fájlok használatával. A fejlesztők behúzhatják a felhasználói felület összetevőit egy vászonra, és testre szabhatják tulajdonságaikat a Properties panelen.

Az Android Studio legfontosabb eleme a beépített emulátor, amely lehetővé teszi a fejlesztők számára, hogy teszteljék alkalmazásaikat virtuális Android-eszközökön, különböző képernyőmérettel, felbontással és Android-verziókkal. Támogatja továbbá a hibakeresést fejlesztői számítógéphez USB-n keresztül csatlakoztatott fizikai Android-eszközökön. A fejlesztők beállíthatnak töréspontokat, ellenőrizhetik a változókat, és valós időben elemezhetik az alkalmazások teljesítményét.

Az Andorid Studio-hoz tartozik továbbá egy Gradle rendszer, ami lehetővé teszi a fejlesztők számára, hogy egyedi build konfigurációkat és függőségeket határozzanak meg projektjeikhez.

Mivel általánosságban elmondható, hogy a világon jelenleg az Android felhasználók vannak többségben, ezért ragaszkodtam ehhez a megoldáshoz, más operációs rendszerek helyett.

Java

A Java egy széles körben használt, magas szintű objektum-orientált programozási nyelv, amelyet a Sun Microsystems fejlesztett ki az 1990-es évek közepén.

A Java szintaxis hasonló a C nyelvekhez, így az ezeket ismerő fejlesztők viszonylag könnyen megtanulhatják.

A Java egy átfogó könyvtárral rendelkezik. Java Development Kit (JDK) néven ismert, amely számos segítséget kínál különféle feladatokhoz. A könyvtár segít a fejlesztőknek robusztus és funkciókban gazdag alkalmazások hatékonyabb felépítésében.

Ezek mellett széles közössége és támogatottsága miatt a telefonos applikáció fejlesztésénél számomra egyértelmű volt a Java használata.

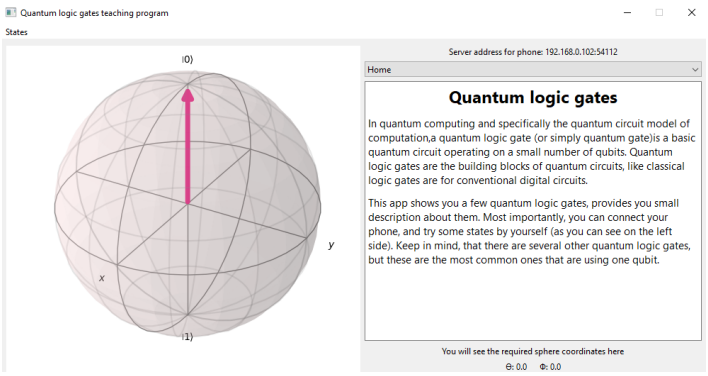
3.3. Asztali alkalmazás bemutatása

A már említett elképzelések alapján alakítottam ki a program kinézetét és funkcióit, amelyek tovább bővíthetők az életciklusa során.

3.3.1. Grafikus felhasználói felület

Fontos szempont volt, hogy mivel az asztali alkalmazás elsősorban inkább bevezető jelleggel rendelkezik a kvantumkapukhoz, így maga a felület és az információk is letisztult megjelenést képviseljenek.

Az ablak megjelenítéséért a Window osztály konstruktora, valamint a hozzá tartozó, többnyire privát metódusok felelnek. A Window osztály örökli a QMainWindow osztályt, amivel hozzáférhetünk a Pyside grafikai elemeihez és azok funkcióihoz.



3.1. ábra. Az asztali alkalmazás kinézete indításkor

Menü

A program tartalmaz egy felső menü sávot, ebben egy States menüponttal. Ezen belül találhatóak az egyes bázisvektor állítási lehetőségek a Bloch gömbhöz.

kód 3.1. Menü inicializálása

```
1 # Menu Bar
2 self.menu = self.menuBar()
3 self.menu_states = self.menu.addMenu("States")
4 self.menu_random = QAction("Random Bloch state", self)
5 self.zero_state_option = QAction("|0> bloch state", self)
```

```
6 self.one_state_option = QAction("|1> bloch state", self)
```

Az összes menüpont kattintással használható. A menü a menuBar, a bázisvektorok menüpontjai pedig a QAction osztályok példányai. A QAction osztály felel a gombok kattintásra történő működéséért, meghívja az adott menüponthoz tartozó eljárást, amennyiben az kiválasztásra került.

kód 3.2. Menü elemek bekötése

```
1 # Menu Bar
2 self.menu_random.triggered.connect(self.__random_state)
3 self.zero_state_option.triggered.connect(self.__zero_state)
4 self.one_state_option.triggered.connect(self.__one_state)
5 self.menu_states.addAction(self.menu_random)
6 self.menu_states.addAction(self.zero_state_option)
7 self.menu_states.addAction(self.one_state_option)
8 self.menu.addMenu(self.menu_states)
```

Bloch gömb

A Bloch gömb megvalósítása a Qiskit visualization csomagjának segítségével történt. Ez tartalmaz egy plot függvényt, ami egy matplotlib ábrával tér vissza.A programban paraméterei közé tartozik egy három elemű tömb, ami alapján a gömb készül, valamint a koordináta típusa.

Az elkészült matplotlib ábrát a Pyside FigureCanvasQTAgg osztály egy példányához adom, ezzel egy "vásznat" biztosítva a Bloch gömbnek. Ez a futás során folyamatosan frissül a különböző vektorállapotok miatt.

A vásznat egy QHBoxLayout példányhoz adom, ami horizontálisan jeleníti meg az elemeket.

kód 3.3. A Bloch gömb kezdő állapota a konstruktorban

```
1 # Bloch sphere part
2 self.bloch_vector = BlochVector.BlochVector(0.0, 0.0)
3 self.fig = plot_bloch_vector([1, self.bloch_vector.theta, self.
    ↪ bloch_vector.phi], coord_type='spherical')
4 self.canvas = FigureCanvasQTAgg(self.fig)
```

Jobb oldali információ panel

A Bloch gömb mellett a másik legfontosabb rész a felületen.

A panel legfelső részén a szerver ip címe látható, ami a QLabel osztály példánya. A címet az általam létrehozott Server osztály egy példányából nyerem ki.

Ez alatt egy legördülő menü helyezkedik el. A QComboBox példányához hozzáadtam a lehetséges kapu változatokat, illetve egy kezdőoldalt. A menüpont változását

folyamatosan ellenőrzi a program, ehhez tartozik egy publikus eljárás, valamint egy változó is. A metódus adott opció alapján alakítja ki a felületet, ez felelős azért, hogy adott kapuhoz a megfelelő információk jelenjenek meg. A változó az ezt követő gombok működése miatt fontos. Az eljárás publikussága a kezdeti állapot beállítás miatt szükséges.

A következő elem a leírás. Ez egy QTextEdit példány, ami tartalmazza az adott kapuk információit. Menüpont váltás miatt folyamatosan cserélődik.

A leíráshoz tartozik egy áramkörrajz a kapuról. Ez az alkalmazás indulásakor nem látszódik, csak amennyiben ellépünk a kezdő oldalról. A rajzot külön generáltattam és lementettem a Qiskit QuantumCircuit csomagjával.

Az áramkör rajzhoz hasonlóan két gomb sem látszódik a kezdő állapotban, amik a QPushButton példányai. Mindkettő függ attól, hogy melyik kapu van kiválasztva. Az első felel az animációs ablak megjelenítéséért. A második gomb lenyomásával a program alkalmazza a kiválasztott kaput a Bloch gömbön és elindít egy forgatási metódust, amennyiben a felhasználó csatlakoztatott telefont. Ha ez nem történt meg, a program egy felugró ablakban figyelmeztet, hogy csatlakoztasson telefont. Az ablak egy QMessageBox példánnyal valósul meg

A panel legalsó részén két QLabel példány látható, amik tartalmazzák a gömb jelenlegi koordinátáit, valamint a forgatás során megjelennek a célkoordináták is.

A felsorolt elemeket egy QVBoxLayout példányhoz adom, ami vertikálisan jeleníti meg őket. Ezt pedig a már említett QHBoxLayout példányhoz adom, hogy a Bloch gömb mellett jelenjen meg.

kód 3.4. A jobb oldali panel és elemeinek létrehozása a konstruktorban

```
1  # Right side of the program
2  self.addr_label = QLabel()
3  self.gates_combo = QComboBox()
4  self.info_text = QTextEdit()
5  self.label = QLabel()
6  self.pixmap = QPixmap()
7  self.show_anim = QPushButton()
8  self.start_measure_button = QPushButton()
9  self.coordinate_label = QLabel()
10 self.current_coordinate_label = QLabel()
11 self.right_side = QVBoxLayout()
12 self.which_gate = ""
```

kód 3.5. A jobb oldali panel elemeinek értékadása

```
1  # Right side of the program
2  self.addr_label.setText("Server address for phone: " + str(
       ↪ server_start.host) + ":" + str(server_start.port))
3
```

```
4  self.gates_combo.addItems(["Home", "Identity", "Pauli-X", "Pauli-Y", "
       ↪ Pauli-Z", "Hadamard", "Phase", "T"])
5
6  self.label.setPixmap(self.pixmap)
7
8  self.coordinate_label.setText("You will see the required sphere
       ↪ coordinates here")
9  self.current_coordinate_label.setText("\u03F4: " + str(self.
       ↪ bloch_vector.theta) + "\t\u03A6: " + str(self.bloch_vector.phi))
10
11 self.info_text.setReadOnly(True)
12 self.label.setVisible(False)
13 self.show_anim.setVisible(False)
14 self.start_measure_button.setVisible(False)
```

kód 3.6. Elemek hozzáadása a jobb oldali panelhez, gombok beállítása

```
1  self.right_side.addWidget(self.addr_label)
2  self.right_side.addWidget(self.gates_combo)
3  self.right_side.addWidget(self.info_text)
4  self.right_side.addWidget(self.label)
5  self.right_side.addWidget(self.show_anim)
6  self.right_side.addWidget(self.start_measure_button)
7  self.right_side.addWidget(self.coordinate_label)
8  self.right_side.addWidget(self.current_coordinate_label)
9  self.right_side.setAlignment(self.addr_label, Qt.AlignCenter)
10 self.right_side.setAlignment(self.label, Qt.AlignCenter)
11 self.right_side.setAlignment(self.coordinate_label, Qt.AlignCenter)
12 self.right_side.setAlignment(self.current_coordinate_label,
13         Qt.AlignCenter)
14
15 self.gates_combo.currentTextChanged.connect(self.gates_combo_options)
16 self.show_anim.clicked.connect(self.__open_anim_window)
17 self.start_measure_button.clicked.connect(self.__start_gate_check)
```

Animációs ablak

A különböző kapukhoz a felhasználó megtekintheti azok animációját is, amely a $|0\rangle$ standard bázisvektorból indul. Ezeket külön kigeneráltattam és kimentettem a Qiskit visualization csomag segítségével.

Az animáció folyamatosan ismétlődik, amíg a felhasználó be nem zárja a megjelent plusz ablakot, amit a hozzá tartozó gomb lenyomásával érhet el. Kattintáskor egy privát metódus indul, ahol a kiválasztott kapu alapján a program példányosít egy általam létrehozott AnimationWindow osztályt. Az osztály konstruktora a megfelelő gif elérési útvonalát tartalmazza, valamint az ablak szükséges beállításait, illetve egy

privát eljárást. A metódus példányosít egy QMovie osztályt, ami a gif megjelenítéséért felel.

kód 3.7. AnimationWindow osztály

```
class AnimationWindow(QMainWindow):
        def __init__(self, gate, parent=None):
                QMainWindow.__init__(self, parent)

                self.setWindowTitle("Gate animation")
                self.setFixedSize(500, 500)

                self.gif_label = QLabel(self)
                self.gif_label.setFixedSize(500, 500)

                self.gif_path = gate

                self.__display_gif()

        def __display_gif(self):
                movie = QMovie(self.gif_path)
                self.gif_label.setMovie(movie)
                movie.start()
```

3.3.2. Kommunikáció a telefonnal

Az asztali alkalmazás legérdekesebb része a telefonnal történő vektorforgatás lehetősége. Ehhez szükséges volt a kapcsolat kialakítása egy telefonos applikáció és a program közt.

A hálózat tevékenységeit az általam létrehozott Server osztály egy példánya kezeli. Konstruktora tárolja a host ip címet, valamint a port számot. A host a számítógép címe, a port számot pedig az operációs rendszer választja ki a szabadok közül.

kód 3.8. A Server osztály konstruktora

```
def __init__(self, HOST, PORT):
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.host = HOST
        self.port = PORT
        self.s.bind((self.host, self.port))
        self.port = self.s.getsockname()[1]
        self.conn = ""
        self.addr = ""
```

Szerver indítása és kapcsolat várása

A Window osztály példányosítása előtt a program elindít egy metódust, amiben egy PhoneConnector osztály példánya foglal helyet.

A példány threading segítségével a háttérben elindítja egyetlen metódusát. Ennek köszönhetően a szerver elindul és várja a telefon csatlakozását. Egy felugró ablak jelzi ennek sikerességét, amit pedig egy Signal példány hív meg. Egyszerre csak egy telefont tud csatlakoztatni a felhasználó, ezt a Server osztály példányának metódusa ellenőrzi.

kód 3.9. A Window osztály példányosítása előtt elindított metódus

```
1 def connect_phone():
2         phone_connector = PhoneConnector()
3         phone_connector.connected.connect(show_phone_connected_message
              ↪ )
4         phone_thread = threading.Thread(target=phone_connector.run)
5         phone_thread.start()
```

kód 3.10. PhoneConnector osztály

```
1 class PhoneConnector(QObject):
2         connected = Signal()
3
4         def run(self):
5                 if server_start.hosting():
6                         self.connected.emit()
```

kód 3.11. A csatlakozást váró függvény a Server osztályban

```
1 def hosting(self):
2         self.s.listen(0)
3         self.conn, self.addr = self.s.accept()
4         return True
```

Szenzor adatok fogadása és tárolása

A telefon szenzor adatait az asztali alkalmazás a forgatás pillanatában kezdi el fogadni és egy tömbben tárolni.

Mivel a telefon már a csatlakozás pillanatában is próbálja küldeni az adatokat, így az első csomag amit fogad a program, eldobásra kerül, csakúgy, mint azok az adatok, amik nem két elemű float tömböt adnak. A felhasználó ezeket az adatvesztéseket nem érzékeli, esetleg a vektor forgatásának lassabb folyamata látszódhat.

Amennyiben a kapcsolat megszűnik, a program üres csomagot fog kapni. Ebben az esetben a szerver bontja a kapcsolatot a klienssel. Amint ez megtörténik, a szerver újra elkezdi várni a kapcsolatot és kezdődik az elejéről a folyamat.

kód 3.12. Szenzor adatok fogadása és tárolása valamint kapcsolat bontása

```
def get_data(self):
        data = self.conn.recv(1024).decode()
        if not data:
                self.conn.close()
                return None
        else:
                angles = data.split(',')
                return angles
```

3.3.3. Matematikai számítások

Ugyan a felhasználó nem érzékeli, de a program jelentős része háttérszámításokon alapul. Ehhez kialakítottam egy BlochVector osztályt, ahol az adott gömbkoordináták alapján tud az alkalmazás számításokat végezni.

Gömbkoordináták

A bejövő szenzor adatok és a számítások egyszerűsítése miatt gömb koordináta-rendszert használtam. Amennyiben ezt nem adom meg a Bloch gömb rajzolásakor, a Qiskit automatikusan a másik opciót, azaz a Descartes-féle koordináta rendszert használja. Ennek megfelelően a paraméterében a három elemű tömb tartalma rendre (ami a 3.3. kódban is látható):

1. Az origótól mért távolság, tehát a gömb sugara. Ez jelen esetben mindig egy lesz. Jele: r

2. Azimut szög, azaz a pont és a $\phi = 0$ tengely által meghatározott sík, valamint $\theta = 0$ sík közötti hajlásszög az $x - y$ koordinátasíkban. Jele: θ

3. A pontot és az origót összekötő egyenes hajlásszöge a $\phi = 0$ irányhoz, azaz az z tengellyel bezárt szög. Jele: ϕ

Mivel a gömb sugara nem változik, így a továbbiakban csak a θ és a ϕ értékei változnak.

A Qiskit csomagja mellett létrehoztam egy BlochVector osztályt, mely konstruktorában tárolja a szükséges értékeket a későbbi számításokhoz, megjelenítésekhez.

Standard- és véletlenszerű bázisvektorok generálása

Az alkalmazás menüsávjában a felhasználó kiválaszthatja, hogy milyen bázisvektor állást szeretne magának a gömb forgatása előtt használni. A kiválasztott opció elindítja az ahhoz tartozó eljárást, ami beállítja a Bloch gömb állapotát.

A véletlenszerű generálás esetén a metódus θ értékét 0 és π között, ϕ -t pedig 0 és 2π közt határozza meg, a gömbkoordináta szabályainak megfelelően.

Az említett szabályok alapján a $|0\rangle$ állapot $\theta = 0$ $\phi = 0$, $|1\rangle$ pedig $\theta = \pi$ $\phi = 0$ állapotot fog kapni.

A megadott koordinátákon kívül nincs különbség a három eljárásban, viszont a gombok és egyéb hívások miatt átláthatóbb opció volt ezeket nem egy metódusban tartani.

kód 3.13. Véletlenszerű bázisvektor generálása és Bloch gömb rajzolása

```
1  def __random_state(self):
2          # Set Bloch sphere to a random state
3          self.bloch_vector.theta = random.uniform(0.0, np.pi)
4          self.bloch_vector.phi = random.uniform(0.0, 2 * np.pi)
5          plt.close()
6          self.fig.canvas.flush_events()
7          self.fig = plot_bloch_vector([1, self.bloch_vector.theta, self
               ↪ .bloch_vector.phi],
8          coord_type='spherical')
9          self.canvas.figure = self.fig
10         self.fig.set_canvas(self.canvas)
11         self.canvas.draw()
12         self.current_coordinate_label.setText("\u03F4: " + str(self.
               ↪ bloch_vector.theta)
13         + "\t\u03A6: " + str(self.bloch_vector.phi))
```

Bázisvektor mátrix kialakítása gömbkoordináták alapján

A kigenerált bázisvektorral a program forgatás kezdésekor kezd el foglalkozni. Ekkor a BlochVector osztálynak egy olyan privát függvénye is lefut, ahol a már ismert θ és ϕ értékeiből kiszámítjuk α és β értékeit, amit egy mátrixban tárolunk. Mivel a bázisvektor egy qubit állapotát jelenti, így a 2.1., valamint a 3.1. képletekből indulhatunk ki.

$$|\Psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle\,, \tag{3.1}$$

ahol i komplex szám, θ és ϕ pedig a már említett gömbkoordináták.

kód 3.14. Bázisvektor mátrix kialakítása a képletek alapján

```
1  def __start_state_vector(self):
2          alpha = np.cos(self.theta / 2)
3          beta = np.exp(1j * self.phi) * np.sin(self.theta / 2)
4          state_vector = np.array([[alpha], [beta]])
5          return state_vector
```

Kapuk alkalmazása a bázisvektor mátrixán

A kapott bázisvektor mátrixon szorzást hajt végre a program, az alapján, hogy melyik kapu lesz kiválasztva. Ahogy azt a 2.2.4. fejezetben láttuk, mindegyik kvantumkapu egyedi mátrix-val rendelkezik. A szorzás által kapott mátrix lesz az új bázisvektor mátrixa.

kód 3.15. Pauli-X kapu alkalmazása a bázisvektorra

```
1 def pauli_x(self):
2         pauli_x_matrix = np.array([[0, 1], [1, 0]])
3         state_vector_after_gate = np.dot(self.__start_state_vector().T
              ↪ , pauli_x_matrix)
4         coordinates = get_spherical_coordinates_from_state_vector(
              ↪ state_vector_after_gate)
5         self.theta = coordinates[0]
6         self.phi = coordinates[1]
```

Új bázisvektor számítása

A 2.1. képlet felírható polár alakban:

$$|\Psi\rangle = (r_\alpha)e^i(\phi a)\,|0\rangle + (r_\beta)e^i(\phi b)\,|1\rangle \tag{3.2}$$

Ebben a formában négy változó van, két valós szám és két képzetes. Ezekből kiindulva az alábbi kód alapján történik théta és fí visszafejtése az új bázisvektorból. Ezeket az adatokat küldi tovább a program a forgatás ellenőrzéséhez.

kód 3.16. Fí és théta kinyerése az új bázisvektorból

```
1  def get_spherical_coordinates_from_state_vector(
       ↪ state_vector_after_gate):
2          alpha_real = state_vector_after_gate[0][0].real
3          alpha_imag = state_vector_after_gate[0][0].imag
4          beta_real = state_vector_after_gate[0][1].real
5          beta_imag = state_vector_after_gate[0][1].imag
6
7          if alpha_real != 0.0:
8                  if alpha_real < 0 < alpha_imag:
9                          alpha_theta = np.arctan(alpha_imag /
                               ↪ alpha_real) + np.pi
10
11                 elif alpha_real < 0 and alpha_imag < 0:
12                         alpha_theta = np.arctan(alpha_imag /
                               ↪ alpha_real) + np.pi
13                 else:
14                         alpha_theta = np.arctan(alpha_imag /
                               ↪ alpha_real)
```

```
15          else:
16                  alpha_theta = 0.0
17
18          r_alpha = np.sqrt((alpha_real ** 2) + (alpha_imag ** 2))
19
20          if beta_real != 0.0:
21                  if beta_real < 0 < beta_imag:
22                          beta_theta = np.arctan(beta_imag / beta_real)
                               ↪ + np.pi
23
24                  elif beta_real < 0 and beta_imag < 0:
25                          beta_theta = np.arctan(beta_imag / beta_real)
                               ↪ + np.pi
26                  else:
27                          beta_theta = np.arctan(beta_imag / beta_real)
28          else:
29                  beta_theta = 0.0
30
31          theta = 2 * np.arccos(r_alpha)
32          phi = beta_theta - alpha_theta
33          if phi < 0:
34          phi += 2 * np.pi
35          coordinates = [theta, phi]
36          return coordinates
```

3.3.4. Vektor forgatása és ellenőrzése

Az előző alfejezetekben tárgyalt elemei az asztali alkalmazásnak, mind hozzásegítenek ehhez a funkcióhoz. A csatlakoztatott telefonnal a felhasználó alkalmazhat egy kaput a kvantumbiten. Amint a háttérszámítások alapján előáll az új bázisvektor, elkezdődik a forgatás, amiért egy privát metódus felel, ahol egy ciklus fut az új állapot eléréséig, vagy hibáig.

Tárolt szenzor adatok feldolgozása

A ciklus folyamatosan kéri az új szenzor adatokat. Ha üres tömböt kap, egy felugró ablak üzenete után a gömböt $|0\rangle$ állapotba állítja, újra kapcsolatot vár és kilép a ciklusból, ezzel együtt az eljárásból is.

Ellenkező esetben a kapott adatokat ellenőrzi. Ha a tömb nem két elemű, valamint nem lehet float típusúvá konvertálni, akkor az adatot "eldobja", folytatja a következő érkezett adattal.

Ezután a beérkezett szenzor adatok közül ϕ értékét módosítjuk, hogy 0 és 2π érték között legyen.

kód 3.17. Tárolt szenzor adatok feldolgozása

```
while True:
        angles = server_start.get_data()
        if angles is None:
                show_message("Phone disconnected! Please reconnect to
                   ↪ continue!")
                self.__zero_state()
                connect_phone()
                break
        else:
                try:
                        angles = np.array(angles, dtype=float)
                        if angles[1] < 0:
                                angles[1] = angles[1] + 2 * np.pi
```

Bloch gömb újrarajzolása

Minden új adatnál, a gömb újrarajzolásra kerül. Az előtte megjelenített rajz, valamint a vászon törlésre kerül, memória- és helyfelszabadítás miatt. Helyére egy új gömb jut, ahol θ és ϕ értéke az új szenzoradatokra cserélődik.

kód 3.18. Bloch gömb újrarajzolása

```
plt.close()
self.fig.canvas.flush_events()
self.fig = plot_bloch_vector([1, angles[0], angles[1]], coord_type='
   ↪ spherical')
self.canvas.figure = self.fig
self.fig.set_canvas(self.canvas)
self.canvas.draw()
```

Gömbkoordináta ellenőrzése

A Bloch gömb kirajzoltatása után a ciklus végén ellenőrzi a program, hogy a jelenlegi vektor állása megfelel-e a kvantumkapu alkalmazásával kapott bázisvektornak. Az egyszerűbb működés érdekében ez 0.1 relatív- és 0.15 abszolút toleranciával működik. Ha a felhasználó ezeken belül van, a program egy felugró ablakban jelzi hogy megtalálta a keresett bázisvektort és a ciklus leáll. Ezután újra elvégezheti a műveletet, de akár más kaput is választhat, mindaddig, amíg a telefon csatlakoztatva marad.

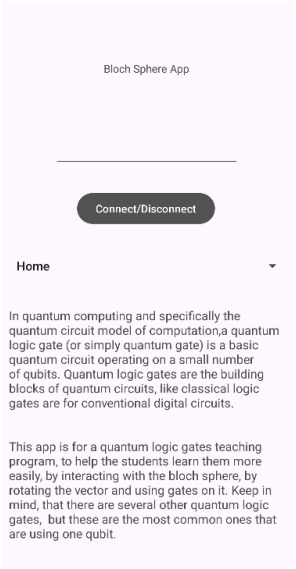
3.4. Applikáció bemutatása

Mivel a hangsúly elsősorban az asztali alkalmazáson van, így az android applikáció egy egyszerű megvalósítást képvisel, viszont hasonlóan az elképzelések alapján készült,

valamint bővíthető.

3.4.1. Grafikus felhasználói felület

A grafikai elemek megjelenítéséért elsődlegesen egy xml fájl felel, amit az Android Studio automatikusan létrehoz az egyes activity-khez. Itt adhatjuk meg az egyes elemeket, akár kódként, de lehetséges a Design panelen behúzni őket az applikáció felületére.



3.2. ábra. Az applikáció kinézete indításkor

Az applikáció tartalmaz egy EditText elemet, ahova a felhasználó a csatlakozási címet írhatja be. Ez csak akkor elérhető, ha a telefon még nem csatlakozott az asztali alkalmazáshoz.

Ehhez tartozik egy gomb, ami a csatlakozást/lecsatlakozást kezeli. Ez befolyásolja többek között az EditText elem elérhetőségét is, de elsősorban nem ez a célja.

kód 3.19. Button létrehozása xml-ben

```
<Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:backgroundTint="#535252"
        android:text="Connect/Disconnect"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.358" />
```

Mivel fontos volt, hogy az asztali alkalmazáson elért rövid információk elérhetőek legyenek a telefonon is, ezért ezt egy Spinner, TextView és egy ImageView elemmel valósítottam meg. Előbbihez az activity-hez tartozó java fájlban hozzáadtam egy tömböt, ami tartalmazza az egyes menüpontokat. Tartozik még ehhez továbbá egy Listener eljárás, ahol az applikáció folyamatosan figyeli, hogy a felhasználó mikor vált menüpontot és ennek megfelelően változtatja a felületet. Az ImageView a főmenüben nem látszik, kapu kiválasztáskor jelenik meg. A leírást tartalmazó elemhez hasonlóan a felület legfelső részén elhelyezkedő cím is TextView elem.

Az xml-ben felsorolt elemek a cím TextView-n kívül a java fájlban példányosítva vannak, az azokkal történő műveletek miatt. Ugyanitt fontos megjegyezni, hogy az applikáció nem lép alvó módba. Ezt egy flag-vel értem el, ami az activity ablakához adódik, egy konstans értékkel.

3.4.2. Kommunikáció az asztali alkalmazással

A felhasználó, az EditText elemben megadhatja a csatlakozási címet, majd gomb lenyomásával próbálhat csatlakozni az asztali alkalmazáshoz. Amennyiben hibás adatot ad meg, ezt az applikáció felugró üzenetben jelzi és javíthatja. Ellenkező esetben tárolom a megadott cím adatait, és egy Socket példány segítségével csatlakoztatom a telefont. Ahhoz, hogy továbbítani tudjam a szenzor adatokat az asztali alkalmazás számára, létrehoztam egy PrintWriter példányt.

kód 3.20. Csatlakozás az asztali alkalmazáshoz

```
1  if (!clicked)
2  {
3          clicked = true;
4          String[] connectiondata = ip_port.getText().toString().split("
               ↪ :");
5          try {
6                  socket = new Socket();
7                  socket.connect(new InetSocketAddress(connectiondata
                       ↪ [0],Integer.parseInt(connectiondata[1])),100);
8                  outToServer = new PrintWriter(new OutputStreamWriter(
                       ↪ socket.getOutputStream()));
9          }
10         catch (IOException e) {
11                 Toast.makeText(MainActivity.this, "Check your
                       ↪ connections!",
12                 Toast.LENGTH_LONG).show();
13                 clicked = false;
14                 return;
15         }
16         catch (ArrayIndexOutOfBoundsException |
               ↪ IllegalArgumentException e){
```

```
17                  Toast.makeText(MainActivity.this, "Wrong address!",
18                  Toast.LENGTH_LONG).show();
19                  clicked = false;
20                  return;
21          }
```

Ha a felhasználó bontani szeretné a kapcsolatot az asztali alkalmazással, akkor azt gomb nyomással teheti meg, vagy bezárja az applikációt. Ilyenkor a fennmaradt, továbbításra váró adatok valamint a szenzor törlésre kerülnek, a kapcsolatot pedig zárja a program. Az applikáció alaphelyzetbe áll, újra lehet csatlakozni.

kód 3.21. Kapcsolat bontása az asztali alkalmazással

```
1  else
2  {
3          clicked = false;
4          outToServer.flush();
5          sensorManager.unregisterListener(MainActivity.this);
6          try {
7                  socket.close();
8          } catch (IOException e) {
9                  Toast.makeText(MainActivity.this, "Can't disconnect!
                       ↪ Check your connections and try again!",
10                 Toast.LENGTH_LONG).show();
11         }
12         ip_port.setEnabled(true);
13 }
```

3.4.3. Szenzor adatok gyűjtése

Miután a telefon sikeresen csatlakozott az asztali alkalmazáshoz, az applikáció példányosít egy SensorManager osztályt, ez felel a szenzorok létrehozásáért, valamint fel- és leregisztrálásáért.

Miután a példányosítás megtörtént, a program létrehoz egy szenzort és regisztrálja azt. Az Android Studio számos szenzortípussal szolgál, ezek mind különböző feladatot látnak el. A forgatás szempontjából számunkra az Orientation szenzor típus a megfelelő. Regisztráláskor fontos megadni a szenzoradat késleltetés idejét. Ez lehet beépített konstans érték, de lehet általunk megadott egyedi érték is.

kód 3.22. Szenzor létrehozása és regisztrálása

```
1  sensorManager = (SensorManager) getSystemService(Context.
       ↪ SENSOR_SERVICE);
2  sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION);
3  sensorManager.registerListener(MainActivity.this,sensor,SensorManager.
       ↪ SENSOR_DELAY_NORMAL);
```

A MainActivity osztály implementál egy SensorEventListener osztályt, ami tartalmaz egy publikus eljárást, ahol a szenzor változást figyelhetjük. Ebben a metódusban tároljuk a szükséges θ és ϕ értékét, majd a már említett PrintWriter példánnyal küldjük az adatokat az asztali alkalmazás számára.

kód 3.23. Szenzor adatok gyűjtése és küldése az asztali alkalmazás számára

```
@Override
public void onSensorChanged(SensorEvent event) {
        angles =  event.values[0] + "," +
        event.values[2] * 2;
        outToServer.print(angles);
        outToServer.flush();
}
```

Kapcsolat bontása esetén a SensorManager példány leregisztrálja a szenzort, és a folyamat kezdődik elölről.

3.5. Tesztelések

3.5.1. Unit tesztek

"A unit-teszt, vagy más néven egységteszt, a metódusokat teszteli. Adott paraméterekre ismerjük a metódus visszatérési értékét (vagy mellékhatását). A unit-teszt megvizsgálja, hogy a tényleges visszatérési érték megegyezik-e az elvárttal. Ha igen, sikeres a teszt, egyébként sikertelen. Elvárás, hogy magának a unit-tesztnek ne legyen mellékhatása.

A unit-tesztelést majd minden fejlett programozási környezet (integrated development environment, IDE) támogatja, azaz egyszerű ilyen teszteket írni. A jelentőségüket az adja, hogy a futtatásukat is támogatják, így egy változtatás után csak lefuttatjuk az összes unit-tesztet, ezzel biztosítjuk magunkat, hogy a változás nem okozott hibát. Ezt nevezzük regressziós tesztnek."

Az asztali alkalmazás során fontos volt tesztelni, hogy a bizonyos bázisvektor számítások megfelelőek-e. Mivel a legtöbb kvantumkapu inverze saját maga (pl. 2.17. képlet), így elegendő volt a kapu kétszeri alkalmazásával ellenőrizni, hogy visszakapjuk-e az eredeti állapotot.

kód 3.24. Pauli-X kapu unit teszt

```
    def test_x_gate(self):
                original_theta = self.bloch_vector.theta
                original_phi = self.bloch_vector.phi
                self.bloch_vector.pauli_x()
                self.bloch_vector.pauli_x()
                self.assertAlmostEqual(original_theta, self.
                    ↪ bloch_vector.theta)
```

```
7                   self.assertAlmostEqual(original_phi, self.bloch_vector
                        ↪ .phi)
```

Fontos volt még továbbá a kliens kapcsolódást is ellenőrizni. Ezt a Python mock könyvtárával tettem, ami lehetővé teszi, hogy a tesztelés alatt álló program részeit fiktív objektumokra cserélje.

Ezek ellenőrzött módon szimulálják a valós objektumok viselkedését. Különösen hasznosak olyan kódok tesztelésekor, amelyek olyan külső rendszerektől vagy erőforrásoktól függenek, amik kiszámíthatatlanabbak, ilyenek például adatbázisok, webszolgáltatások vagy hardvereszközök.

kód 3.25. Kliens csatlakozás unit teszt

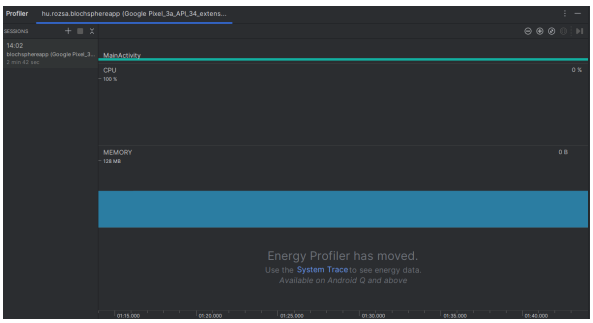
```
1  class TestServer(unittest.TestCase):
2          def test_server_binding(self):
3                  HOST = '127.0.0.1'
4                  PORT = 12345
5
6                  # Mocking the socket object
7                  with patch('socket.socket') as mock_socket:
8                  server = Server(HOST, PORT)
9
10                 # Assert that the socket was called with the correct
                       ↪ arguments
11                 mock_socket.assert_called_once_with(socket.AF_INET,
                       ↪ socket.SOCK_STREAM)
12
13                 # Mock the bind method of the socket object
14                 mock_socket_instance = mock_socket.return_value
15                 mock_socket_instance.bind.assert_called_once_with((
                       ↪ HOST, PORT))
16
17                 # Mock the listen method of the socket object
18                 mock_socket_instance.listen.return_value = None
19
20                 # Mock the accept method to return a dummy connection
21                 dummy_conn = MagicMock()
22                 dummy_addr = ('127.0.0.1', 12345)
23                 mock_socket_instance.accept.return_value = (dummy_conn
                       ↪ , dummy_addr)
24
25                 # Test the hosting method
26                 self.assertTrue(server.hosting())
```

3.5.2. Teljesítmény teszt-Android Studio Profiler

Az Android Studio egyik erőssége a már felsoroltak közül, a beépített profiler funkció. Segít a fejlesztőknek Android alkalmazásaik teljesítményének elemzésében és optimalizálásában, valamint a felhasználói élmény javításában. Különféle lehetőségeket kínál, többek közt CPU, memóra, energia és hálózat valós idejű figyelését, miközben az alkalmazás emulátoron vagy fizikai eszközön fut.

Metódusköveteése lehetővé teszi, hogy részletes információkat gyűjtsön a metódusok végrehajtásáról az alkalmazásban. Ez hasznos lehet a teljesítményproblémák azonosításához és a kódvégrehajtás optimalizálásához.

A Profiler különféle vizualizációs eszközöket, például diagramokat és grafikonokat biztosít, amelyek segítenek a teljesítményadatok hatékonyabb értelmezésében és elemzésében.



3.3. ábra. Profiler az Android Studioban

3.5.3. Felhasználói tesztek

Általánosságban elmondható, hogy a tesztelés során elfogultak lehetnek a fejlesztők, vagy esetleg nem gondolnak bizonyos esetekre. Nem lehet számítani minden eshetőségre és viselkedésre. Emiatt a diplomamunkám megosztottam szaktársaimmal és olyan emberekkel is, akik úgymond laikusan állnak a témakörhöz. Ennek oka, hogy az asztali alkalmazás célcsoportja elég széleskörű, lényegében bárki, aki érdeklődik a témakör iránt és szeretne betekintést nyerni.

A visszajelzések alapján képes voltam olyan eseteket javítani, amik nem merültek volna fel, ha csak magam tesztelem az applikációt és az asztali alkalmazást.

3.6. Továbbfejlesztési lehetőségek

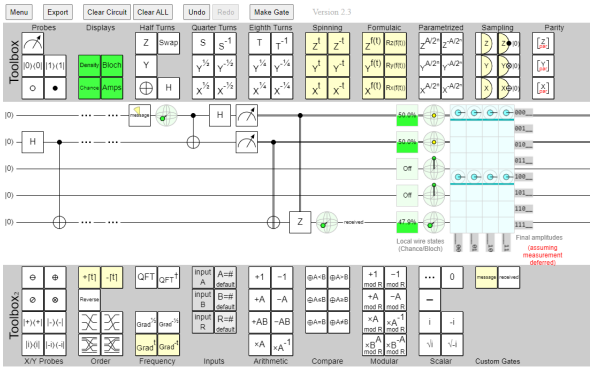
Egyik program vagy applikáció sem lehet teljesen kész és hibátlan. Mindig keletkeznek új elképzelések, amik részben, de az is lehet, hogy jelentősen megváltoztatják koncepciót.

3.6.1. Részletes matematikai egyenletek mutatása

A program jelenleg mindent a háttérben számít, a felhasználó semmit nem érzékel ebből. Egy külön textbox-ban meg lehetne jeleníteni az adott qubit egyenletét, valamint annak változását különböző állapotokra, kapukra.

3.6.2. Saját áramkör kialakítása

A legtöbb kvantumkapuval foglalkozó szimulátort sajátosan kialakítható áramkörök szimulálására tervezték. Ezek általában behúzható kapukat tartalmaznak, mindegyik mellé rövid leírás társul. A felhasználó így tetszőleges kvantumáramköröket alakíthat ki, ezzel még jobban érzékelve a kvantumbitek reakcióját bizonyos kapuk alkalmazására.



3.4. ábra. Quirk, az egyik legismertebb webes szimulátor

A 3.4. ábrán látható szimulátorhoz hasonlóan a programba is beemelhető lehet egy hasonló megvalósítás, mivel a Qiskit a saját áramkör kialakításban és szimulálásban a legerősebb. Ez ötvözhető lehet a forgatás ellenőrzéssel, amennyiben az áramkörök kialakítását egy kvantumbitre és kapukra korlátozzuk.

3.6.3. Több kvantumbites kapuk

Jelen szakdolgozat ugyan csak egy qubites kapukkal foglalkozik, felmerült hogy későbbiekben ez kibővüljön több kvantumbitre is. Ezek a kapuk ugyanúgy megkapnák azokat az animációkat és pontos ismertetőket.

Ennek a lehetőségnek kihívása az, hogy a jelenleg elérhető Bloch gömb vizualizációkkal körülményes egy olyan forgatás ellenőrzést létrehozni, mint ahogy azt az egy kvantumbites változatoknál láttuk.

Viszont ettől függetlenül a Qiskit lehetőséget nyújt a több kvantumbites kapuk áramkörben történő megjelenítésére és szimulálására. Ezt egy részletes leírással ötvözve szintén egyszerűbb megértést nyújthat a kvantumkapukhoz.

3.6.4. Diagramok

A Qiskit erősségei közé tartoznak még a szimuláció során kinyerhető adatok, amiket diagramok formájában is megjeleníthetünk. Amennyiben ezt ötvözzük a tetszőleges áramkörök lehetőségével, akkor képes lehet a felhasználó látni, hogy bizonyos helyzetekben hogyan reagálnak a kvantumbitek.

Forgatás szempontjából ez az opció kifejezetten memóriaigényes, mivel csakúgy mint a programban lévő Bloch gömb, a diagramok is folyamatos újrarajzolást igényelnének minden új beérkező szenzor adat után.

3.6.5. Könyvtárak frissen tartása

Ahogy a kvantuminformatika, úgy az ezzel foglalkozó Python könyvtárak is rohamosan fejlődnek. A Qiskit jelenleg az 1.0 verziójánál tart és folyamatosan újabb lehetőségekkel, megoldásokkal bővítik könyvtárukat. Ebből adódóan bármikor érkezhet olyan megoldás, ami hozzátenne a programhoz, esetleg hatékonyabbá tenné azt.

3.6.6. IOS applikáció

Az applikáció jelenleg csak androidos telefonokra elérhető, viszont az IOS lehetőséggel több emberhez el tudna jutni a program. Nehézsége, hogy IOS applikációkat MacOSX környezetben lehet csak fejleszteni. Számos megoldás létezik erre a problémára, például a VirtualBox vagy a VMWare, amik virtuális számítógépek kialakítását teszik lehetővé.

Összegzés

Szakdolgozatomban egy olyan összetett témakörrel foglalkoztam, ami egyszerre tartalmaz fizikát, komplex matematikai számításokat és még számos területet. Elsődleges célom volt, hogy egy olyan tanulássegítő lehetőséget fejlesszek, ami több szempontból képes bevonni a felhasználót. Saját célkitűzésem pedig mindenképp az volt, hogy minél többet tanuljak, nem csak a témáról, hanem a választott technológiákról is.

Természetesen a kutatás és a fejlesztés során számos kihívással kellett szembesülnöm, amik megoldásai nagyban segítették a fejlődésem. Sikerült olyan technológiákat jobban megismernem, amikről eddig egyáltalán nem volt hátterem, vagy kifejezetten kevés. Emellett mélyíthettem tudásom fizikai és matematikai szempontból is, amik eddig ezen a szinten jelentősen távol álltak tőlem.

Mivel a kvantuminformatika folyamatosan fejlődik, így ezt a projektet is folytatni szeretném. Számos kiaknázatlan lehetőség van még benne, amennyiben elmozdulna az asztali alkalmazás és az applikáció is a kevésbé bevezető ismertetők irányába. Fontos itt még egyszer megjegyezni, hogy egyik szoftver sem lehet teljesen hibátlan, tehát biztosan található még javítani való is. Valamint a használt technológiák folyamatos fejlődése miatt is kifejezetten fontos a karbantartás.

Mindezek ellenére úgy gondolom sikerült elérnem a kitűzött céljaim a projekttel kapcsolatban és remélhetőleg sikerül még ennek egy olyan irányt vennie, ami helytáll a kvantumkapukkal foglalkozó szimulátorok és tanulássegítő szoftverek közt.

Irodalomjegyzék

Nyilatkozat

Alulírott ....., büntetőjogi felelősségem tudatában kijelentem, hogy az általam benyújtott, ... című szakdolgozat önálló szellemi termékem. Amennyiben mások munkáját felhasználtam, azokra megfelelően hivatkozom, beleértve a nyomtatott és az internetes forrásokat is.

Aláírásommal igazolom, hogy az elektronikusan feltöltött és a papíralapú szakdolgozatom formai és tartalmi szempontból mindenben megegyezik.

Eger, 2021. szeptember 25.

aláírás

A *Nyilatkozatot* kitöltve nyomtassa ki, írja alá,
majd szkennelve tegye ennek a helyére!