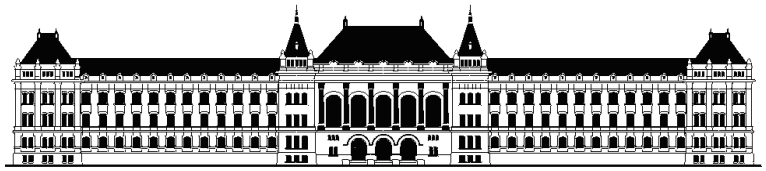


Budapesti Műszaki és Gazdaságtudományi Egyetem
Természettudományi Kar

TDK Dolgozat

Interaktív animáció a kvantumos hibajavításról: a felületi kód

Készítette:
Zanathy Marcell
Budapest XIV. Kerületi Teleki Blanka Gimnázium

Konzulens:
Márton Áron
MSC hallgató
BME Természettudományi Kar

Konzulens:
Asbóth János
Egyetemi docens
BME Elméleti Fizika Tanszék

Kivonat

A kvantuminformatikát kutató fizikusok számára az egyik legnagyobb kihívás egy olyan eljárás megtalálása jelenti, mellyel hatékonyan lehet adatokat tárolni és számításokat végezni egy zajos kvantumszámítógépen. E problémára javasolt egyik legígéretesebb megoldás a *felületi kód* (surface code). Ennek ábrázolása komoly kihívást jelenthet, ugyanis nehéz elképzelni a fizikai kvantumbitek azon erősen összefonódott állapotot, melyben tároljuk a kvantuminformációt.

Dolgozatomban ismertetem a kvantumjelenségek leírásához szükséges fogalmakat és a felületi kód működésének lényeges elemeit. Ezután bemutatom, azt az interaktív animációt, mely a kód összefonódott állapotait jeleníti meg. A programot számomra ismeretlen JavaScript nyelven írtam (annak érdekében, hogy webhelyről is futtatható legyen) a ChatGPT segítségével.

Munkám izgalmas bepillantást jelenthet abba, hogyan lehet bonyolultabb matematikai vagy természettudományos ismereteket igénylő programozási feladatokat elvégezni a ChatGPT segítségével.

Tartalomjegyzék

1. Bevezetés

A kvantuminformatika egy rendkívül sokat tárgyalt területe a kvantumos hibajavítás (quantum error correction). E terület kvantuminformáció hibatűrő tárolásával foglalkozik zajos környezeteben. E kihívásra számos javaslat született, viszont talán egyik javaslat sem bír olyan meggyőző eredményekkel, mint a felületi kód (surface code). (1)

Dolgozatom első részében röviden ismertetem a releváns fogalmakat és összefüggéseket. Bemutatom, hogyan lehet információt kvantummechanikai megfontolások segítségével tárolni, továbbá kitérek a felületi kód működésére és, hogy miért hatékony ez az eljárás kvantuminformáció hibatűrő tárolására.

A második részben ismertetem a felületi kód erősen összefonódott kvantumállapotát megjelenítő általam készített animációs programot. Bemutatom a projektben használt jelöléseket, a program funkcióit, elkészítésének rövid leírását és esetleges felhasználási lehetőségeit.

A kód a ChatGPT (2) asszisztenciájával készült. A dolgozat harmadik részében e munka programozási tapasztalataira térek ki. Az itt leírtak lényegi bepillantást nyújthatnak komolyabb tudományos ismereteket igénylő programozási folyamatokba a ChatGPT-vel és megkönnyíthetik esetlegesen a jövőben készülő hasonló programok létrejöttét.

2. Fontos kvantuminformatikai alapfogalmak

Az alábbiakban ismertetem a kvantumbitek működéséhez szükséges összefüggéseket, valamint a felületi kódhoz tartozó alapvető fogalmakat. (3)

2.1. Kvantumrendszerek matematikája

Bizonyos kvantumjelenségek számítástechnikai alkalmazása jelentős mértékben megkönnyíthet néhány specifikus számítási feladatot (lásd: (4; 5; 6; 7; 8)), illetve a kvantuminformáció hibatűrő tárolását is lehetővé teszik (1). E könnyebbség magyarázata a kvantummechanika alapvető természetében keresendő és abban, hogy a kvantumbitek viselkedésükben lényegukben különböznek a klasszikus bitektől.

A kvantumbitek állapotait kétdimenziós komplex vektorokkal szokás jelölni a következő formában:

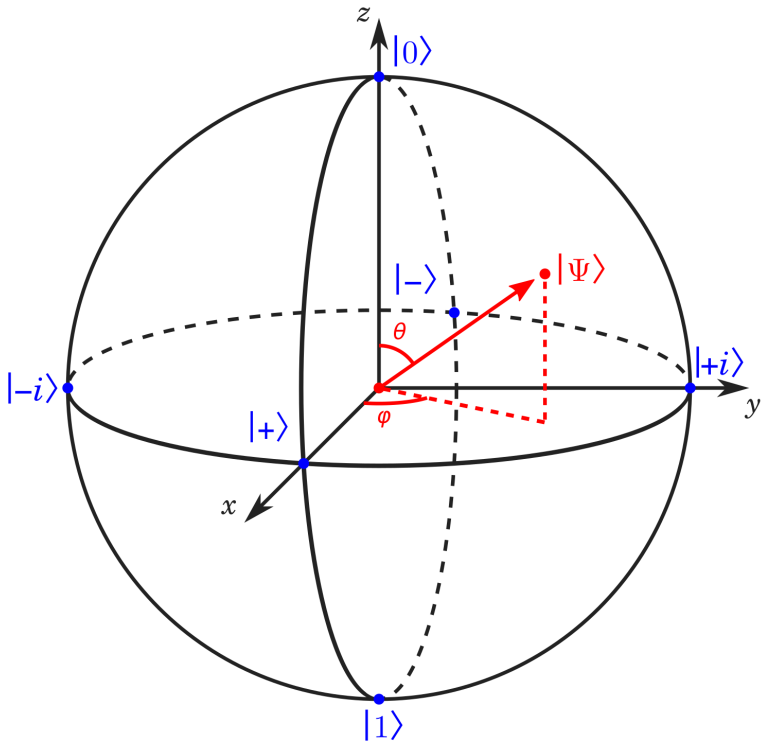
$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,,\ ahol:\ \alpha,\beta \in \mathbb{C}, \quad |\alpha|^2 + |\beta|^2 = 1 \tag{1}$$

Ahol $|0\rangle$ és $|1\rangle$ a számítási bázis két bázisvektora[1].

[1] A két vektor ortonormált.

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2}$$

Geometriai alakban egy kvantumbit állapotát a Bloch-gömb segítségével ábrázolhatjuk(1). A Bloch-gömb egy egységgömb, aminek középpontja az origóban van, és minden pontja egy egyedi kvantumállapotnak felel meg. Az állapotvektorokat a gömb középpontjából kiinduló vektorokként jeleníthetjük meg.



1. ábra. Bloch-gömb. Forrás: (9).

A kvantumbit állapotát a Bloch-gömbön polárkoordinátákkal adhatjuk meg:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle, \tag{3}$$

ahol $\Theta \in [0;\pi]$ és $\phi \in [0;2\pi[$.

Az állapotok közötti transzformációk forgatási műveletekként jelennek meg a gömbön, amelyek matematikailag kvantumkapuk segítségével írhatók

le. Kvantumkapuk a kvantumbitek olyan transzformációi, amik a hagyományos "logikai kapuk" kvantumos megfelelői.

A dolgozat szempontjából fontos transzformációk a Pauli X és a Pauli Z operátorok. A Pauli X operátor a Bloch-gömb x tengelye mentén forgatja el az állapotvektort π szöggel. A Pauli Z operátor pedig a Bloch-gömb z tengelye mentén forgatja el az állapotvektort π szöggel.

A Pauli X és Z operátorok a következő mátrixoknak feleltethetőek meg:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{4}$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{5}$$

Dirac jelöléssel a két Pauli operátor a következő alakban írható fel:

$$X = |1\rangle \langle 0| + |0\rangle \langle 1|, \tag{6}$$

$$Z = |0\rangle \langle 0| - |1\rangle \langle 1| \tag{7}$$

Ezeket az operátorokat lehetséges több kvantumbites rendszerek esetében is definiálni.[2] Egy n kvantumbites rendszerben a Pauli X és Pauli Z operátorok az egyes kvantum biteken a következőképpen fejezhetők ki:

Az i-edik qubiten ható Pauli-X operátor az alábbiak szerint definiálható[3]:

$$X_i = I \otimes I \otimes \cdots \otimes X \otimes \cdots \otimes I \tag{8}$$

Itt X az i-edik helyen van és $n-1$ identitásmátrix van a tenzorszorzatban. A mátrix mérete: $2^n \times 2^n$.

Hasonlóképpen, az i-edik qubiten ható Pauli Z operátor[4]:

$$Z_i = I \otimes I \otimes \cdots \otimes Z \otimes \cdots \otimes I \tag{9}$$

Itt is Z az i-edik helyen van, és $n-1$ identitásmátrix szerepel az összefüggésben. A mátrix mérete $2^n \times 2^n$.

Amennyiben egy kvantumrendszeren méréseket szeretnénk végezni (megmérni egy operátort), a mérésünk eredménye $+1$ és -1 lehet.

Annak a valószínűsége, hogy $+1$ lesz az X operátor mérési eredménye:

$$P_{+1} = \langle \psi | \frac{1+X}{2} | \psi \rangle. \tag{10}$$

[2]Ezt már nem lehet a Bloch-gömbön ábrázolni.

[3]Ha az i-edik qubiten hat a nem triviális operátor, akkor a i-edik tag az X

[4]Ha az i-edik qubiten hat a nem triviális operátor, akkor az i-edik tag a Z

A Pauli Z operátor +1-es mérési eredménye:

$$P_{+1} = \langle\psi|\frac{1+Z}{2}|\psi\rangle. \tag{11}$$

A -1 mérési eredmény valószínűsége a következő Pauli X operátor esetében:

$$P_{-1} = \langle\psi|\frac{1-X}{2}|\psi\rangle. \tag{12}$$

Pauli Z esetében pedig:

$$P_{-1} = \langle\psi|\frac{1-Z}{2}|\psi\rangle. \tag{13}$$

Mindkét esetben igaz, hogy a valószínűségek együttes összege 1 lesz:

$$P_{+1} + P_{-1} = 1 \tag{14}$$

Amennyiben több kvantumbites rendszereken szeretnénk méréseket végezni (akár több kvantumbiten egyszerre), a mérési eredményekhez tartozó valószínűségeket hasonlóképpen számíthatjuk ki. Például (egy kilenc kvantumbites rendszerben) így számolhatjuk ki az X_1X_7 operátor mérési eredményeinek valószínűségeit:

$$P_{+1} = \langle\psi|\frac{1+X_1X_7}{2}|\psi\rangle \tag{15}$$

$$P_{-1} = \langle\psi|\frac{1-X_1X_7}{2}|\psi\rangle \tag{16}$$

A mérési eredmény ismeretében meg tudjuk határozni, hogy milyen állapotban lesz a rendszer a mérés elvégzése után.

Maradva az előző példánál[5], ha a mérési eredmény +1, akkor a rendszer állapota:

$$|\psi\rangle \to N \times \frac{1+X_1X_7}{2}|\psi\rangle \tag{17}$$

Ha a mérési eredmény -1, akkor a rendszer állapota:

$$|\psi\rangle \to N \times \frac{1-X_1X_7}{2}|\psi\rangle \tag{18}$$

Itt N az együtthatók 1-re való normálásáért felelős tényező.

[5]Minden Pauli X és Z operátornál így kell kiszámítani a mérés utáni állapotot.

Az $\frac{1\pm X_1 X_7}{2}$-t gyakran egy projektorként írják fel:

$$\mathbb{P}^{O}_{+1} = \frac{1+O}{2}, \tag{19}$$

ahol O tetszőleges Pauli X vagy Pauli Z operátor (vagy operátorok) szorzata. Egy adott O (X vagy Z) operátorhoz tartozó projektor az az operátor[6], amelyet, ha hattatunk egy tetszőleges $|\psi\rangle$ állapoton, akkor $|\psi\rangle$ abba az állapotba kerül, mintha O-t $|\psi\rangle$-ben mértem volna ugyanabban az értékben.

$$\mathbb{P}^{O}_{\pm 1}|\psi\rangle \equiv N\frac{1\pm O}{2}|\psi\rangle \tag{20}$$

A kvantuminformatikában az egyik legfontosabb jelenség, mely elősegíti kvantuminformáció hibatűrő tárolását zajos környezeteben a kvantumos összefonódás. Egy $|\psi\rangle$ kvantumrendszer összefonódott állapotban van, ha $|\psi\rangle$ állapotát nem lehet a szuperpozíciót alkotó egyedi kvantumbitek tenzorszorzataként (tensor product) meghatározni.

Ezt leggyakrabban a Bell-állapotokkal(10) szokás szemléltetni. Ezek a következők:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{21}$$

$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \tag{22}$$

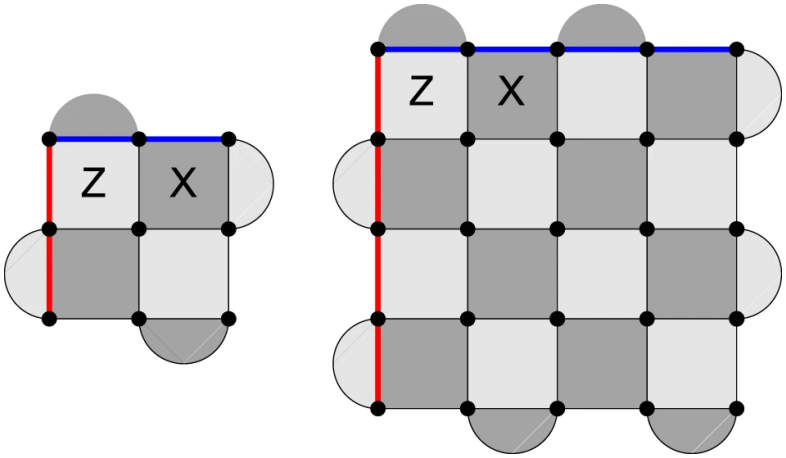
$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \tag{23}$$

$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \tag{24}$$

2.2. A felületi kód

A kvantuminformatikában az egyik leggyakrabban használt hibajavító kód a felületi kód. Ennek többek között az lehet az oka, hogy alapvető természetéből fakadóan alkalmas számos hibatípus kezelésére és úgy definiáljuk rajta stabilizátorokat, hogy csupán egymáshoz közeli kvantumbitekre vonatkozzanak (ezt nevezzük lokális konnektivitásnak). A kódot alkotó kvantumbitek egy $p \times q$-s rácson helyezkednek el (p és q páratlan pozitív egész számok).(11; 12)

[6]Létezik +1-es és −1-es projektor is.



2. ábra. Plaquette (sötészszürke) és vertex (világosszürke) stabilizátorok a felületi kódon. A logikai X operátor a kék, a Z-t pedig piros színnel jelöljük. Forrás: (13).

Annak érdekében, hogy adatokat tudjuk tárolni a felületi kódon, annak egy logikai állapotban kell lennie, mely egy erősen összefonódott kvantumállapot.

$$|\Psi_L\rangle = \alpha|0_L\rangle + \beta|1_L\rangle \tag{25}$$

$|\psi_L\rangle$ logikai állapot, ha

$$\forall S : S|\psi_L\rangle = |\psi_L\rangle, \tag{26}$$

ahol S a felületi kód egy tetszőleges stabilizátora.[7]

Stabilizátoroknak Pauli X és Z operátorok specifikus szorzatait nevezzük[8] (maguk is Pauli operátorok), melyeket két típusba sorolhatunk. Vannak kettő és négy kvantumbiten ható stabilizátorok. A két és a négykvantumbites stabilizátorok esetében beszélhetünk X és Z típusú stabilizátorokról. Négy kvantumbites stabilizátorok esetében az X típusú stabilizátorokat *plaquette*-nek, a Z típusút pedig *vertex*-nek nevezzük.

Annak érdekében, hogy érvényesen definiáljuk a stabilizátorokat, úgy kell azokat meghatározni, hogy kommutáljanak egymással.

Definiálni szoktunk még a felületi kódban logikai X és logikai Z operátorokat is. Ezek ugyanúgy változtatják meg a felületi kód állapotát, mint az egyedi Pauli X és Z operátorok az egykvantumbites rendszerek állapotait.

[7]Ezzel a definícióveg ekvivalens, hogy bármely stabilizátor mérése esetén +1 lesz az eredmény.

[8]Éppen ezért ugyanazok az összefüggések vonatkoznak rájuk, mint a fentebb definiált többkvantumbites Pauli operátorok (hatása, mérése) esetében.

Például, ha egy logikai X operátort hattatjuk a

$$|\psi_L\rangle = \alpha|0_L\rangle + \beta|1_L\rangle \tag{27}$$

állapoton, akkor az eredmény:

$$X_L|\psi_L\rangle = \alpha|1_L\rangle + \beta|0_L\rangle \tag{28}$$

lesz.

Hasonlóképpen:

$$Z_L|\psi_L\rangle = \alpha|0_L\rangle - \beta|1_L\rangle. \tag{29}$$

A logikai operátorok kommutálnak a stabilizátorokkal. Ezt a következő módon írhatjuk le:

$$SO_L = O_L S \tag{30}$$

ahol O_L egy logikai operátor és S a felületi kód egy stabilizátora.

Ezek után könnyen megállapítható, hogyan lehet meghatározni a felületi kód logikai bázisállapotait.

Az összes kvantumbitet $|0\rangle$-ba vagy $|1\rangle$-be inicializáljuk. Ezután hattatjuk rá az összes stabilizátorhoz tartozó +1-es projektort ($\mathbb{P}^S_{+1}$). Végül az együtthatókat 1-re normáljuk.

Egy háromszor hármas felületi kód logikai bázisállapotai például a következők:

$$\begin{aligned}|0_L\rangle = \frac{1}{\sqrt{16}}(&|000000000\rangle + |110110000\rangle + |001001000\rangle + |111111000\rangle \\ &+ |000100100\rangle + |110010100\rangle + |001111100\rangle + |111011100\rangle \\ &+ |000011011\rangle + |110101011\rangle + |001010011\rangle + |111100011\rangle \\ &+ |000111111\rangle + |110001111\rangle + |001110111\rangle + |111000111\rangle)\end{aligned} \tag{31}$$

$$\begin{aligned}|1_L\rangle = \frac{1}{\sqrt{16}}(&|111111111\rangle + |001001111\rangle + |110110111\rangle + |000000111\rangle \\ &+ |111011011\rangle + |001101011\rangle + |110010011\rangle + |000100011\rangle \\ &+ |111100100\rangle + |001010100\rangle + |110101100\rangle + |000011100\rangle \\ &+ |111000000\rangle + |001110000\rangle + |110001000\rangle + |000111000\rangle)\end{aligned} \tag{32}$$

3. Az interaktív animáció

3.1. Az interaktív animáció célja

Mint ahogyan azt már korábban említettem, a felületi kód az egyik legélénkebben tárgyalt eljárás kvantuminformáció hibatűrő tárolására. A felületi

kód működésének egyik legalapvetőbb része, hogy egy kvantumbitnyi információt tárol sok kvantumbit erősen összefonódott állapotában. Ebből következik az, hogy amennyiben vizualizálni szeretnénk a felületi kódot, elengedhetetlenül szükséges egy olyan megjelenési formát találnunk, amiben meg tudjuk jeleníteni sok kvantumbit (jelen esetben háromszor három) összefonódott állapotát. Dolgozatomban ezt mutatom be.

3.2. Az animáció megjelenése

Egy bázisállapot úgy jelenik meg, hogy a megfelelő kvantumbitek felveszik a megfelelő értékeket, azaz a nyilak a képernyőn a megadott bázisvektorhoz igazodva állnak be a megfelelő irányba.

Az animáció a felületi kód állapotát úgy jelenít meg, hogy az állapotban szereplő bázisállapotoknak megfelelő konfigurációkat egymás után, villogtatva mutatja a képernyőn. Egy-egy konfiguráció annyi ideig látszik, amennyi a súlya a szuperpozícióban.

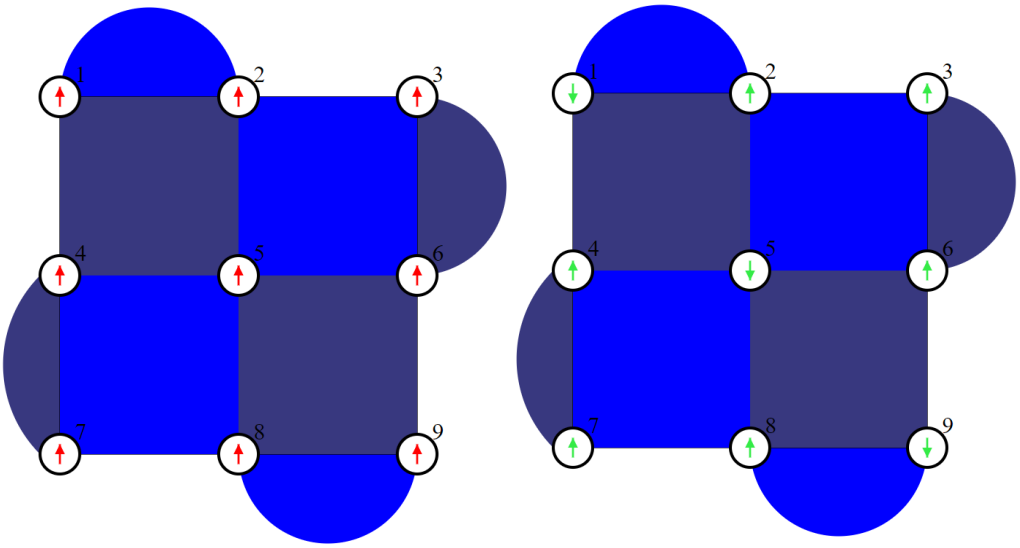
Mind a kvantumállapotot, mind az animáció lefutásának idejét a felhasználó adja meg.

Egy bázisállapot megjelenésének ideje:

$$t = \frac{|c_k|^2 \cdot n}{\sum |c|^2}, \tag{33}$$

ahol c egy bázisállapot együtthatója, c_k a képernyőn pillanatnyilag megjelenített bázisvektirhoz tartozó együttható és n az animáció lefutásának ideje, amelyet a felhasználó ad meg.

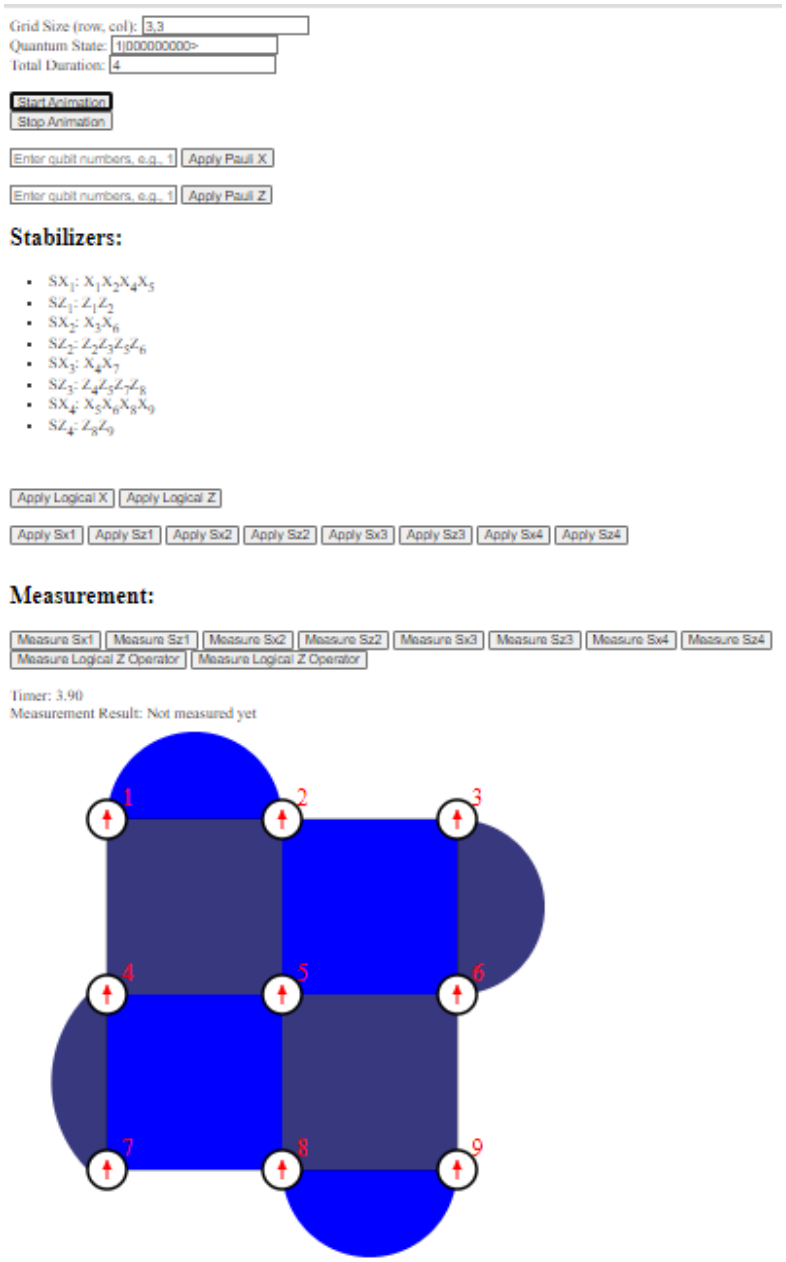
Az animáció (a fentebb tárgyalt módon) képes a felhasználó által megadott kvantumállapotokat megjeleníteni a képernyőn. Amennyiben a felhasználó olyan formában adja meg az állapotot, hogy az tovább egyszerűsíthető, akkor azt a legegyszerűbb formára hozza. A felhasználó képes tetszőleges Pauli X és Z operátorokat és stabilizátorokat hattatni a megjelenített állapoton. A program legfontosabb funkciója a Pauli X és Z operátorok és a stabilizátorok mérése. A felhasználó megmérhet minden Pauli (X és Z) operátort vagy stabilizátort. A környezet hibáit Pauli-operátorok véletlenszerű hattatásának lehet megfeleltetni.



(a) Bázisállapot (000000000|⟩) pozitív relatív fázissal

(b) Bázisállapot (|100010001⟩) pozitív relatív fázissal

3. ábra. A megjelenített kvantumbitek egy bázisállapotban $|0\rangle$ vagy $|1\rangle$ értéket vesz fel. Az animációban minden kvantumbitet egy karika jelenít meg, melyben egy nyíl van. Ez a nyíl a képernyőn felfelé mutat, ha a kvantumbit a $|0\rangle$ és lefelé, ha az $|1\rangle$ állapotban szerepel a szuperpozícióban. A relatív fázist a nyilak színei jelölik. Ha a bázisállapot előjele pozitív, akkor a nyilak piros színnel (HEX: # ff0000) látszanak a képernyőn, negatív előjelű bázisállapotok esetében a nyilak sötétzöld színűek (HEX: # 34eb46).



4. ábra. Az oldal megjelenése, ahonnan elérhető az animáció.

A programot számos potenciális funkcióval lehet még kiegészíteni. Először is, lehetséges megfelelően optimalizálni a kódot (ennek kezdeményei a jelenlegi állapoton is látszanak) tetszőleges számú kvantumbitekre (definiálva hozzá a megfelelő stabilizátorokat). Építhetünk a kódba véletlenszerű hibákat létrehozó fejlesztéseket. Továbbá, hibajavító algoritmust is írhatunk az

animációhoz.

(Az animáció ide kattintva érhető el).

3.3. Az animáció elkészítésének menete

A folyamatot, ahogyan az animáció készült, jól elkülöníthető módon három különböző részre lehet osztani. Az animációt javascript nyelven, a ChatGPT-4 asszisztenciájával készült.

A programozási nyelv kiválasztásánál az volt a szempont, hogy az animáció külső weblapról is elérhető legyen. Ez ugyan előnye a JavaScript nyelvnek, hátránya azonban, hogy mind nekem, mind témavezetőimnek ismeretlen (volt). A nyelv elsajátítása helyett úgy döntöttünk, kísérletet teszünk a javascript közvetett programozására, a ChatGPT segítségével. Ilyen módon a munka nagy része a ChatGPT munkájának irányítása (promptok) és a javascript programAnnak minőségellenőrzése volt. Az animációk elkészítése során nagyban támaszkodtam a D3.js (Data-Driven Documents) JavaScript könyvtárra, mely alkalmas komplex, interaktív és könnyen testreszabható animációk elkészítésére.

A munkafolyamat három részét egyértelműen meghatározta, hogy a ChatGPT milyen mértékben képes a megadott feladatokat elvégezni.[9]

Az első részben maga a felületi kódot megjelenítő rács készült el és azok a programrészek, melyek a felhasználó által megadott kvantumállapotot jelenítik meg a képernyőn. Ez a rész volt a munka legkönnyebb és leggyorsabban végrehajtható része. A ChatGPT megértett szinte minden utasítást („promptot")[10] és a szakkifejezések felhasználásával sem volt probléma. A második részben az operátorok és stabilizátorok hattatásáért felelős részek készültek el. Az erre vonatkozó feladatokat is viszonylag könnyedén megoldotta a ChatGPT, bár itt már jóval több hibát kellett javítani. A harmadik rész a mérésekért felelős programrészek megírása volt. Szemben az első két résszel, ebben a részben a ChatGPT szinte semmilyen feladatot nem tudott önállóan megoldani. Ebben a programrészben számottevően nagyobb részt kellett teljesen önállóan megírni, és csak bizonyos függvények logikai felépítésének felvázolására volt alkalmas a ChatGPT.

Felmerülhet a kérdés: miért okozhatott a ChatGPT-nek lényegesen nagyobb nehézséget a mérésért felelős kódrész megírása? Ugyan erre nem áll módomban egzakt választ adni, viszont két lehetőséget tartok elképzelhetőnek (nem kizárt, hogy mindkettő egyaránt szerepet játszott). Az első az, hogy a meglévő kód terjedelme akkortájt érte el azt a terjedelmet, miután már nem

[9] Természetesen néhány kisebb változtatás e részektől függetlenül került a kódba.

[10] Más kontextusban e szónak sok különböző jelentése van.

lehetett egyetlen prompta bemásolni az egészet. Lehetséges, hogy a program terjedelme is ekkorra érte el azt a szintet, miután a ChatGPT már képtelen volt organikus módon tovább frissíteni azt. A második lehetőség, hogy a mérésért felelős kódnak (a többi részhez képest) jelentősen bonyolultabb számításokat kellett volna elvégezni (skalárszorzat kiszámítás, projektorok hattatás stb.). Plauzibilis magyarázatnak tartom, hogy a ChatGPT jelenlegi állapotában még alkalmatlan ennyire komplex számítások elvégzére alkalmas kódot írni.

3.4. Felhasználási lehetőségek

Néhány terület, ahol az animáció hasznos lehet:

- A felületi kód bonyolult matematikai struktúrájának vizualizációja hozzájárulhat annak mélyreható, intuitív megértéséhez.
- A felületi kód vizualizációja hozzásegíthet a hibajavító algoritmusok hatékonyabb implementálásához és teszteléséhez.
- Az interaktív animáció hozzásegíthet ahhoz, hogy a felületi kód működésének oktatása egyszerűbb legyen, ugyanis mivel ennek az interaktív animációnak a segítségével könnyebben meg lehet érteni annak legalapvetőbb működéséti elveit.
- A vizualizáció hozzásegíthet ahhoz, hogy más tudományterületek képviselői is könnyebben megérthessék annak működését. Ezzel elősegítve számos interdiszciplináris kutatást. A legújabb kvantuminformatikai eredmények ugyanis nem kizárólag a fizika számára lehetnek relevánsak.[11]

4. Programozási tapasztalatok a ChatGPT-vel

Mint azt már korábban említettem, az animációs program megírásában nagymértékben támaszkodtam a ChatGPT-re. Ez részben megkönnyítette a programozás folyamatát és komoly útmutatásul szolgált munkám során. A továbbiakban az ezzel kapcsolatos tapasztalataimat írom le, illetve igyekszem pontos képet alkotni a ChatGPT jelenlegi lehetőségeiről.

A ChatGPT az OpenAI mesterséges intelligencia kutató laboratórium által kifejlesztett chatbot, mely a felhasználókkal való folyamatos kommunikáció automatizálása során értelmezőmodelleket használ, melyek segítségével

[11] Erről tanúskodik, hogy hány különböző tudományterületre kiterjednek a kvantuminformatika felhasználási módjai.

a bevitt információkat azonnal interaktívan kezeli. Viszont felhasználási lehetőségei túlmutatnak az egyszerű kérdés-válasz dinamikán vagy a szöveg generáláson.

A program segítséget nyújthat nagyszámú programozással kapcsolatos rutinfeladat elvégzésében vagy kezdő programozók számára (mely kategóriába én is beletartozom), akik nem rendelkeznek kellő előismeretekkel bizonyos feladatokhoz. A ChatGPT a dolgozat szempontjából leglényegesebb funkciója, hogy képes egy bonyolultabb kód vázlatát megadni, melyet a kellő ismeretek birtokában a kód optimális állapotáig lehet finomítani.

Fontos megjegyzés, hogy a fejezetben leírtak a ChatGPT fejlettségének arra az állapotára vonatkoznak, melyben a dolgozat megírásának ideje alatt volt. Tekintve, hogy a ChatGPT jövőbeli lehetőségeivel és képességeivel kapcsolatos vélemények megoszlanak, jelen dolgozatban erre vonatkozóan semmilyen megállapítást nem tennék a mostani állapotokból kiindulva.

4.1. Prompt engineering: módszertani megjegyzések

Mielőtt még ismertetném a konkrét tapasztalatokat, szeretnék néhány módszertani javaslatot tenni azzal kapcsolatban, hogyan lehet általánosságban a ChatGPT-vel programozni. Mivel a ChatGPT-t elsődlegesen angol nyelvű utasításokhoz fejlesztik, ezáltal e nyelvet érti a leginkább. A legjobb eredmény elérése érdekében célszerű angol nyelven kommunikálni a programmal (ügyelve a nyelv (többé-kevésbé) helyes használatára).

A ChatGPT jelen állapotában nem alkalmas arra, hogy minden esetben hibátlan utasításokat adjon, gyakorta szükségünk lesz a megírt programban manuálisan javítani a kódon. Továbbá, hosszabb programrészek esetén gyakorta rövidíti le a válaszokat (sok esetben nem is egyértelműen használja a rövidítéseket). Hogy elkerüljük az ebből származó nehézségeket, szükséges legalább alapszinten megtanulni a használt programozási nyelvet. Így megfelelően be lehet illeszteni az új kódrészeket vagy esetleges frissítéseket a meglévő programba.

Lehetőség szerint érdemes lehet a ChatGPT standard, ingyenesen elérhető változata (ChatGPT-3.5) helyett a ChatGPT-4-et használni olyan feladatokra, melyek ténylegesen összetettebbek és részletességük okán nagyobb elővigyázatosságot igényelnek. A tapasztalatok azt mutatják, számos munkaóra spórolható meg ezzel a lépéssel, továbbá a válaszok is gyakorta lényegesen színvonalasabbak, mint a standard (ChatGPT-3.5) esetében.

4.2. Hogyan lehet hatékony utasításokat adni a ChatGPT-nek?

A megadott promptokat egyértelmű és kellően specifikus nyelvezettel írjuk meg, biztosítva hozzá a megfelelő kontextust. Lehetőség szerint tegyünk kikötést arra vonatkozóan, hogy milyen jellegű választ várunk (például: magyarázd el a kvantumos összefonódást középiskolás szinten.).

Célszerű már a legelején megadni, milyen programozási nyelvben szeretnénk megíratni a kódot, illetve, hogy milyen további eszközöket szeretnénk felhasználni (például jelen esetben a JavaScript és a D3.js). Ezeket az utasításokat későbbi válaszok során is többnyire be fogja tartani a program. Azokban az esetekben, amikor mégis eltérne e feltételektől, a korábbi parancs szövegébe fogalmazzuk bele azt, ami a válaszból hiányzik és generáljuk újra a választ az új szöveggel. Az esetek túlnyomó többségében ez megoldja a felmerülő problémákat.[12]

Amennyiben a leendő programunknak bonyolult matematikai műveletekkel vagy természettudományos törvényszerűségekkel kell dolgoznia, legelőször érdemes lehet különböző analógiák vagy példák helyett az utasításban felhasználni megfelelő szakkifejezéseket, annak érdekében, hogy a program működése minél jobban igazodjon a kívánt dolog működéséhez, jellemzőihez. Ha a ChatGPT képes ez alapján értelmezni a megadott utasítást (és minden más hiba lehetőségét is kizárhatjuk), nagy eséllyel kaphatunk szinte tökéletes kivitelezést. Amennyiben a válasz mégis hibás lenne, adjunk meg néhány a témára vonatkozó képletet vagy összefüggést, mely képes lehet megértetni a ChatGPT-vel a feladatot.

Fontos viszont megjegyezni, hogy miután tisztáztuk, pontosan milyen dolgokat kell a leendő programnak megjelenítenie és már a ChatGPT megírta a kód releváns részét, későbbi módosítások során érdemes lehet a programozás kifejezéseinek szintjén folytatni a kommunikációt esetleges (például, nem azt mondjuk, hogy a kvantumállapotot módosítsa, hanem közvetlenül arra a listára referálunk, amiben a kvantumállapotot tároljuk) további frissítések, javítások során. E lépéshez természetesen elengedhetetlen az adott programozási nyelv minimális ismerete.

Például az egyik legelső prompt, amit a ChatGPT-nek adtam a következő volt:

[12]Érdemes már a munkafolyamat legelején meghatározni (a lehető legnagyobb pontossággal) a felhasználandó eszközöket.

‹ 4 / 4 ›

MA

Write an animation in JavaScript using D3.js. Put everything into one unified code, with all quantum bits. The animation will show a surface code in an entangled quantum state. The surface code will consist of an arbitrary number of quantum bits. Let the user give the number of the quantum bits in every row and columm. An entangled quantum state consist of an arbitrary number of basis states and there are corresponding coefficents to each basis state. For examole: 1/√2|111000111> + 1/√2|000111000> + 1/√2|100010001>. Let the user give the entangled quantum state in the following format: give one unified text-container for the coefficents and basis states alike. Note that the sum of the square of the absolute values of the coefficents should be 1 (|a|^2+|b|^2+|c|^2+...+|d|^2=1 - it is a law of quantum mechanics). If it is not the case warm the user with a text on the screen saying „The coefficents are incorrect" – it's color should be red. It is possible that some coefficents correspondinging to a basis state will have a negative (-) value. Separate each basis state with a „+" or a „-" sign depending on the value of the coefficent. When the user separates two basis vectors with a „-" sign it means that the coefficent of the basis vector is negative. The user can increase the number of basis vectors with wrigting new basis vectors and coefficents into the text container. Here are a few examples how the quantum states should look like: 1/√2|111000111> + 1/√2|000111000>; 1/√2|111000111> - 1/√2|000111000>. Note that it is possible that all the basis vectors are in one parenthesis with just only one coefficient. For example: 1/√2(|111000111> + |000111000>). In these cases all the basis vectors have the same coefficent but if they have a negative sign inside the parenthesis then the basis vectors coefficient is negative (for example: 1/√2(|111000111> - |000111000>), in this case the first basis vector should be represented with a positive coefficent and the second one should be represented with a negative coefficient).
The quantum bits will be represented as circles on the screen and each quantum bit will be connected with straight lines with black color if they are in one row or column. Mark each quantum bit from 1 to 9 and show it on the screen above and left to the quantum bit. The numbering goes left to right vertically in each raw. When the quantum bit is in 0 the circle it is represented with is white. If the quantum bit is in 1 then the circle it is represented with is either red (#FF0000) or aqua (#00FFFF) depending on wether the basis vector it is represented is positive or negative.

5. ábra. A promptban szakkifejezések és a felhasználandó programozási eszközök (nyelv, könyvtárak stb.) és példák szerepelenk. [Az utasítás az animáció egy régebbi változatához készült.]

A jelenlegi projektben azt kértem a ChatGPT-től, hogy egy összefonódott állapotban (entangled quantum state) lévő felületi kódot (surface code) jelenítsen meg a program. Fontos viszont megjegyezni, hogy ez a módszer sem működik minden esetben. Például a ChatGPT képtelen volt kizárólag az alapján megírni a Pauli-operátorok és stabilizátorok méréséért felelős részt, hogy elmagyaráztam (példák segítségével), hogyan lehet kiszámítani az adott mérés eredményeit. Ehhez a feladathoz lényeges manuális módosításokat kellett végezni a programon belül.

Azokban az esetekben, amikor a ChatGPT nem tudja a megadott szakkifejezések vagy esetleges matematikai vagy természettudományos magyarázat alapján megérteni a feladatot, a legjobb megoldás lehet analógiákhoz folyamodni, azaz *egyszerűbb szinten kommunikálni* a programmal.

Így jártam el a projektorok hattatására vonatkozó programrész elkészítésében. A promptokban lineáris algebra helyett az az utasítást adtam, hogy tároljon el egy listában egy olyan állapotot, melyet a következőképpen lehet megkapni: $\frac{1}{2}$(a+b) vagy $\frac{1}{2}$(a-b), ahol a az eredeti, felhasználó által megadott állapot és b pedig az az állapot amelyiken hattatuk a megfelelő Pauli operátort vagy stabilizátort.

A fenti példa esetében a következő uatsítást adhatjuk:

"*Calcualte the following state: c=a+b where a is the displayState [az a lista, melyben a képernyőn mejelenő állapotot tárolja a program] and b is the displayState (a) after applying the X1 operator to it. Then count the number of different basis vectors in c. If some basis vector cancels each other out in this new state (a+b) then do not count it, and you have identical basis vector in the new state (a+b) count it as many times as they appear in the new state (a+b). After this you get the first number (c) you have to work with. Then divide this number (c) by the number of basis vectors in a and then divide this result with 2. You will get a number (dotP) between 0 and 1. Next: determine the result of the measurement (+1 or -1). The probability of +1 as a result is dotP and -1 as a result is 1-dotP. The new state after the measurement is a+b (simplified and normalised) if the measurement result was +1. And if the measurement result was -1 then the new state is a-b (where a is the displayState and b is the displayState (a) after applying the X1 operator to it).*"[13]

Hogy leellenőrizzük, megfelelő utasításokat adtunk-e a programnak, érdemes lehet tisztázó kérdéseket feltenni a program számára és nem kizárólag a konkrét program megírására vonatkozó promptokat írni. Például egy konkrét feladaton keresztül könnyen kideríthetjük, érthető volt-e a kiadott utasítás (a megírt program logikája szerint milyen eredményt kapunk és milyen gondolatmenet alapján jutott el a válaszig, vagy általánosságban magyarázza el az új kódrész működését, funkcióját). Amennyiben e kérdésre rossz válasz érkezik, nevezzük meg a hiba forrásait és a program javítani fogja.[14] A megadott példák között legyenek egyértelmű és bonyolultabb esetek egyaránt.

Az előző példa esetében feltehetünk például egy ilyen üzenetet: "*What happens, when the user gives the following state: |000111000>. What will be the measurement result and the measured state? Detail your calculation.*"

Felmerülhet az is, hogy tisztában vagyunk azzal, hogy egy adott feladatot nagy vonalakban hogyan lehet elvégezni (például egy adott érték kiszámításához milyen képletet vagy összefüggést kell felhasználnunk (ténylegesen

[13] A gondolatmenettől független okokból végül nem ez a megvalósítás került a programba.

[14] Sok esetben a hiba nem a megírt programban van, hanem magában a tisztázó üzenetben (mely kideríthető a kapott válaszokból).

képlet alakban megadva), természetesen ügyelve az egyértelműségre). Konkrétabban: milyen logika vagy programozási elemek segítségével kell egy adott kódrészt megírni, milyen bővítmények használatával, a stb. Ilyenkor érdemes a parancsba ezt is beleírni, hogy miképpen hajtsa végre a ChatGPT a feladatot.

Például annak a függvénynek a megíratásához, mely egy megadott kvantumállapotot egyszerűsít le javaslatként megadhatjuk azt, hogy hozzon létre egy listát az azonos együtthatójú bázisállapotokból és egyszerűsítse le.

Lásd: "*The siplification can be done in the following way: the program checks if there are basis vectors in the displaied quantum state which are the same, then the program checks the coefficients in front of them. The program adds all of those coefficients together.*

For example you can create a function that takes an array of terms like ['+5x', '-3x', '+4y', '+2y', '-1z', '+3z', '+2w'] and simplifies it to ['+2x', '+6y', '+2z', '+2w']."

Mikor megfogalmazunk egy üzenetet, érdemes törekedni a rövidítések hanyagolására, és mindenhol a lehető legegyértelműbben és legtisztábban fogalmazni. Ez azért szükséges, mert a ChatGPT félreértheti azokat a rövidítéseket, amiket a természetes emberi kommunikáció során használhatunk. (Erre jó példa lehet az ötödik ábrán5 szereplő utasítás szövege).

Ha különösen fontos számunkra egy adott programrész részletes ismerete, kérjük, hogy a program kommentjeiben részletezze a megértendő rész pontos működését, feladatát.[15]

Az előző ponthoz hasonlóan kérhetünk tanácsokat arra vonatkozóan is, milyen programozási nyelvet vagy könyvtárakat használjunk egy bizonyos feladat elvégzésére. Adjuk meg, milyen kritériumokat kell figyelembe venni az adott projekt megvalósításában (például, hogy külső weboldalról is futtatható legyen vagy bonyolult számításokat kell gyorsan elvégeznie).

4.3. Nagyobb projektrészekre vonatkozó promptok

Felmerülhet a kérdés, hogyan lehet a leghatékonyabb módon megírni egy lényeges és komplex programozási feladathoz kapcsolódó promptot vagy promptokat? Erre a kérdésre sajnos nincsen egyértelmű válasz. A leghatékonyabb eljárást (vagy eljárásokat) a konkrét feladat alapján állapíthatjuk meg. Alapvetően három egymástól jól megkülönböztethető eljárást tartok lehetségesnek.

[15] A ChatGPT automatikusan is ír kommenteket a teljes kód bizonyos részleteihez olyan módon, e kommentek alapján is lehetséges (legalább nagyvonalakban) a kód működésének megértése.

Ezzel kapcsolatban két fontos megjegyzés. Először is e három technikáti használhatunk egymással párhuzamosan. Másodszor, érdemes megadni a meglévő kódot vagy releváns kódrészt a ChatGPT-nek (bár a promptok hossza esetén van felső karakterlimit) abban az esetben, ha (1) egy jelentősebb fejlesztést szeretnénk beilleszteni a már meglévő programba vagy (2) lényegi módosításokat végeztünk a kódban a ChatGPT tudta nélkül[16].

4.3.1. Hosszú üzenetek

Az első elképzelés az, hogy az adott bonyolult feladatot egyetlen (vagy csak nagyon kevés) részletes és átfogó utasításként adjuk meg, mely a teljes feladatot ismerteti hiányosságok nélkül.

Jó példa lehet erre az ötödik ábrán5 lévő utasítás.

Ennek az az előnye, hogy világos, átfogó, minden szempontra kiterjed és potenciálisan felgyorsíthatja a fejlesztés menetét, ugyanis nem kell több utasításon keresztül részletezni a feladatot. Hátránya viszont, hogy a ChatGPT nehezen tudja feldolgozni a hosszú utasításokat és számos fontos részlet kimaradhat a megírt kódból.[17] Továbbá, a megadott utasítások utólag nehezen módosíthatóak. Ráadásul hibás utasítás esetén könnyedén az egész kódra kiterjedő problémákkal találkozhatunk.

4.3.2. Projekt alkotórészekre bontása logiaki felépítettség alapján

A második lehetőség, hogy a feladatot több kisebb utasításra bontjuk fel, melyek a projekt logikai felépítése szerint követik egymást a megfelelő sorrendben.

Például az ötödik ábrán5 lévő üzenetet felbonthatjuk több rövidebb és egyszerűbb promptra:

1. Write an animation in JavaScript using D3.js. The animation shows a surface code in an entangled quantum state. The surface code will consist of an arbitrary number of quantum bits. Let the user give the number of the quantum bits in every row and columm.

2. An entangled quantum state consist of an arbitrary number of basis states and there are corresponding coefficents to each basis state. For example: $1/\sqrt{2}|111000111> +1/\sqrt{2}|000111000> +1/\sqrt{2}|100010001>$.

[16] A kódot egy fejlesztéstől függetlenül is megadhatjuk ellenőrzés vagy hibák kijavíttatásának céljából.

[17] Általában a hosszabb promptok esetén számos kisebb-nagyobb részlet kimarad melyeket utólag kell korrigálni. Ennek ellenére még így is gyorsabb lehet ez a megközelítés, mint a többi, mert még így is kevesebb promptot kell írni.

It is possible that some coefficents correspondinging to a basis state will have a negative (-) value. Separate each basis state with a „+” or a „-” sign depending on the value of the coefficent.

3. The quantum bits will be represented as circles on the screen and each quantum bit will be connected with straight lines with black color if they are in one row or column. Mark each quantum bit from 1 to 9 and show it on the screen above and left to the quantum bit. The numbering goes left to right vertically in each raw. When the quantum bit is in 0 the circle it is represented with is white.

4. *Et cetera*

Ennek a módszernek komoly előnye, hogy jóval rugalmasabb, mint az első. A kód sokkal egyszerűbben módosítható esetleges változások esetében, illetve a meglévő funkciók megfelelő működése sokkal könnyebben ellenőrizhető. Továbbá, nagyobb eséllyel kaphatunk megfelelően megírt programot egy ilyen rövidebb prompt után. E módszernek komoly negatívuma, hogy a rövid utasítások nem adnak teljes képet a végcélról és lehetséges, hogy nem áll össze a ChatGPT-nek a pontos feladat. Ez a későbbiekben komoly tervezési problémákhoz is vezethet. Ráadásul sokkal több időt vesz igénybe, mint, ha nagyobb utasításokon keresztül kommunikálnánk a programmal.

4.3.3. Elemi funkciókra vonatkozó promptok

A harmadik megoldás hasonló a másodikhoz. Egy feladatot ugyanúgy több rövidebb üzenet alapján íratunk meg. Viszont a részüzenetek nem a konkrét projektben betöltött szerepük alapján íródnak meg, hanem elemi funkcióik figyelembevételével. Ez különösen akkor lehet hasznos, ha a ChatGPT nem tud pusztán tudományos szakkifejezések alapján megírni egy adott kódrészt.

Például a mérést a következő elemi utasításokkal is megírhatjuk:

1. Create a new array in the code outside of any function for a new quantum state. This new state $(a + b)$ is composed of the basis vectors and coefficients of the `displayState` (a) and the original state in which an $X1$ operator is applied (b). Write this new state to the screen. Then multiply this new state by 1/2. Finally, use the `simplifyAndNormalize` function on it.

2. Now write a function that calculates the following: count the number of basis vectors in the `newstate`, then divide it by the numbers in the `displayState`. And then divide this whole result by 2. And store this state as a variable called `dotP`.

3. Then you should write a function that creates an array for the so-called measurement state. The measurement state is either the `combinedStateObjects` (1st way) or a different state that can be obtained by subtracting a version of the `displayState` in which the $X1$ operator is applied from the `displayState` (2nd way). The way the program can decide which way to use is probabilistical. The probability of the 1st way is `dotP`, and the probability of the 2nd way is $1-$`dotP`. Note that `dotP`$+(1-$`dotP`$) = 1$.

Ennek a megközelítésnek az az előnye, hogy kód részei egymástól függetlenül (akár több különböző beszélgetésben) fejleszthetők és javíthatók, ezáltal csökkentve a feladat elkészülésének idejét. Továbbá, könnyebb új részeket hozzáadni a meglévő programhoz. Viszont, komoly árnyoldala ennek a kivitelezésnek, hogy a független részek nem feltétlenül lesznek kompatibilisek egymással (csak manuális változtatások segítségével). Ezen kívül, fennáll annak a veszélye, hogy kontextus hiányában fontos funkciók veszhetnek el.

4.4. Hogyan korrigáljuk a meglévő program hibáit?

A ChatGPT-vel való programozás nem csak olyan utasításokból áll, melyben a meglévő kódba újabb frissítések kerülnek. Az esetek döntő többségében ki kell javítani a megírt programban lévő hibákat. A következőkben azt fogom kifejteni, hogyan lehet ezeket megtenni.

Először is, mielőtt még lefuttatnánk a megírt kódot, érdemes még a ChatGPT által válaszként adott szövegdobozban leelenőrizni azt, ugyanis sokszor ez alapján előre meg tudjuk mondani, hogy az adott program működik-e vagy sem. Ennek az az oka, hogy (főként bonyolult utasításoknál) a ChatGPT gyakran csak vázlatosan ír meg bizonyos részeket. Mikor ilyennel találkozunk, érdemes újabb promptokkal újraíratni ezeket a részeket.

```
function applyPauliOperator(gridType) {
    // Logic for applying Pauli operators
    // E.g. check which operator is being applied and update quantum state and visual representation accordingly
}

function applyStabilizer(gridType) {
    // Logic for applying stabilizers
    // E.g. check which stabilizer is being applied and update quantum state and visual representation accordingly
}
```

6. ábra. A ChatGPT a két fügvénynek csak a vázát írta meg

Amikor azt tapasztaljuk, hogy a ChatGPT által írt program hibás, az önálló javításon túl lehetőségünk van olyan utasításokat adni, melyekkel ezeket korrigálhatjuk. Ezeknek az utasításoknak azon túl, hogy jelzik a hibát a ChatGPT-nek, ki kell térniük arra is, pontosan mivel van a probléma, ez

alapján írni a javító promptokat. A hibajavítás „hatásfokát" tovább növelhetjük, ha megadjuk, hogy melyik programrészben van a hiba (ez a hiba milyensége alapján könnyen megállapítható).

Amennyiben elakadtunk a sokadik hibajavító prompt után is, fennállhat annak a lehetősége, hogy nem volt megfelelő az a megközelítés, ahogyan a hibát orvosolni kívántuk. Ilyenkor érdemes lehet újraírni a legelső, a hibára irányuló üzenetet és újrakezdeni a javítást. Ha a hibajavító promptot nincs módunk máshogy megírni, újragenerálhatjuk az adott választ. Gyakran ez is elegendő, mert a ChatGPT magától észreveszi a saját hibáit.

Ha a megírt kódban egyszerre több hibát találunk, azokat érdemes egyenként kijavíttatani, ugyanis ellenkező esetben nincs rá garancia, hogy minden probléma orvosolva lesz.

Amikor újabb fejlesztéseket teszünk a programhoz vagy módosítjuk azt, gyakori jelenség, hogy az új részekben a különböző listák adatstruktúrái inkonzisztensek lesznek egymással, mely azt eredményezi, hogy a függvények nem lesznek általánosan használhatóak. A hibáknak ezt a típusát azért emeltem ki külön, mert (tapasztalataim szerint) ez az egyik leggyakoribb és a (ChatGPT segítségével) legnehezebben korrigálható.

Ezen kívül a következő hiba típusok jellemzőek a ChatGPT által írt kódokra: szintaxisok nem megfelelő használata, logikai hibák (például ciklusváltozók nem megfelelő értékeket kapnak), befejezetlen programrészek, inkonzisztens megnevezések használata (például: változók, tömb- és listanevek új neveket kapnak módosítások során).

Mint láthatjuk, ezek a hibák nagyon hasonlítanak azokra a hibákra, amiket más emberek is vétenek. A hibák többnyire nem valami alapvető számítástechnikai tényezőkben keresendők. Éppen ezért ajánlatos saját kezűleg is felülvizsgálni a programot.

5. Konklúzió

Dolgozatoban ismertettem, milyen kvantummechanikai fogalmak szükségesek a felületi kód működésének leírására. Továbbá, ismertettem néhány, a felületi kóddal kapcsolatos alapfogalmat.

Ismertettem a háromszor három kvantumbites felületi kódot megjelenítő animációt. A program annak érdekében, hogy weblapról futtatható legyen, JavaScript nyelven íródott (a D3.js könyvtárat felhasználva) a ChatGPT-t felhasználva.

Az animáció egy felületi kódot jelenít meg egy erősen összefonott állapotban. A felhasználó hattathatja és meg is mérheti a kód Pauli operátorait és stabilizátorait.

A programank számos felhasználási lehetősége van (megkönnyíti a felületi kód működését, elősegíti az interdiszciplináris kutatásokat stb.).

Végül a ChatGPT minőségellenőrzéséből származó tapasztalatokat ismertettem. Kitértem a prompt engineering módszertanára, illetve ismertettem, hogyan lehetséges (komplex) utasításokat hatékonyan megfogalmazni a ChatGPT-nek és hogyan lehet kijavítani azokat.

Hivatkozások

[1] A. Kitaev, „Fault-tolerant quantum computation by anyons," *Annals of Physics*, vol. 303, pp. 2–30, Jan. 2003. https://www.sciencedirect.com/science/article/pii/S0003491602000180.

[2] OpenAI, „OpenAI chat," 2023. https://chat.openai.com/.

[3] E. G. Rieffel and W. H. Polak, *Quantum computing: A gentle introduction*. MIT Press, 2011.

[4] P. W. Shor, „Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[5] M. Schuld, I. Sinayskiy, and F. Petruccione, „An introduction to quantum machine learning," *Contemporary Physics*, vol. 56, no. 2, pp. 172–185, 2015.

[6] N. Gisin, G. Ribordy, W. Tittel, and H. Zbinden, „Quantum cryptography," *Reviews of Modern Physics*, vol. 74, pp. 145–195, mar 2002.

[7] S. Lloyd, „Universal quantum simulators," *Science*, vol. 273, no. 5278, pp. 1073–1078, 1996.

[8] R. P. Feynman, *Feynman lectures on computation*. CRC Press, 2018.

[9] Prefetch, „Understanding the concept of Bloch sphere," 2023. Hozzáférés dátuma: 2023-11-01, https://prefetch.eu/know/concept/bloch-sphere/.

[10] J. S. Bell, „On the Einstein Podolsky Rosen paradox," *Physics Physique Fizika*, vol. 1, no. 3, p. 195, 1964.

[11] A. Pesah, „An interactive introduction to the surface code," 2023. Accessed: 2023-10-30, https://arthurpesah.me/blog/2023-05-13-surface-code/.

[12] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, „Surface codes: Towards practical large-scale quantum computation," *Physical Review A*, vol. 86, Sept. 2012.

[13] S. Bravyi, M. Englbrecht, R. König, and N. Peard, „Correcting coherent errors with surface codes," *npj Quantum Information*, vol. 4, no. 1, p. 55, 2018.

A. A forráskód

```
<!DOCTYPE html>
<html lang="en">
<head>
    <script src="https://d3js.org/d3.v7.min.js"></script>
</head>
<body>
<div>
        <label for="gridSize">Grid Size (row, col):</label>
        <input type="text" id="gridSize" value="3,3">
    </div>
    <div>
        <label for="quantumState">Quantum State:</label>
        <input type="text" id="quantumState" value="1|000000000>">
    </div>
    <div>
        <label for="duration">Total Duration:</label>
        <input type="number" id="duration" value="4">
<br><br>
    <button onclick="startAnimation()">Start Animation</button>
<br>
<button onclick="stopAnimation()">Stop Animation</button>
<br><br>
    <!-- Pauli Operator Buttons -->
    <input type="text" id="Xbits" placeholder="Enter qubit numbers, e.g., 1,2">
<button onclick="applyOperator('X')">Apply Pauli X</button>
<br><br>
    <input type="text" id="Zbits" placeholder="Enter qubit numbers, e.g., 1,2">
<button onclick="applyOperator('Z')">Apply Pauli Z</button>
<h2>Stabilizers:</h2>
<ul>
        <li>SX<sub>1</sub>: X<sub>1</sub>X<sub>2</sub>X<sub>4</sub>X<sub>5</sub></li>
        <li>SZ<sub>1</sub>: Z<sub>1</sub>Z<sub>2</sub></li>
        <li>SX<sub>2</sub>: X<sub>3</sub>X<sub>6</sub></li>
        <li>SZ<sub>2</sub>: Z<sub>2</sub>Z<sub>3</sub>Z<sub>5</sub>Z<sub>6</sub></li>
        <li>SX<sub>3</sub>: X<sub>4</sub>X<sub>7</sub></li>
        <li>SZ<sub>3</sub>: Z<sub>4</sub>Z<sub>5</sub>Z<sub>7</sub>Z<sub>8</sub></li>
        <li>SX<sub>4</sub>: X<sub>5</sub>X<sub>6</sub>X<sub>8</sub>X<sub>9</sub></li>
        <li>SZ<sub>4</sub>: Z<sub>8</sub>Z<sub>9</sub></li>
    </ul>

<br><br>

    <!-- Logical X and Z Buttons -->
    <button onclick="applyOperator('LogicalX')">Apply Logical X</button>
    <button onclick="applyOperator('LogicalZ')">Apply Logical Z</button>
```

```
<br><br>

    <!-- Stabilizers Buttons -->
    <button onclick="applyOperator('Sx1')">Apply Sx1</button>
    <button onclick="applyOperator('Sz1')">Apply Sz1</button>
<button onclick="applyOperator('Sx2')">Apply Sx2</button>
<button onclick="applyOperator('Sz2')">Apply Sz2</button>
<button onclick="applyOperator('Sx3')">Apply Sx3</button>
<button onclick="applyOperator('Sz3')">Apply Sz3</button>
<button onclick="applyOperator('Sx4')">Apply Sx4</button>
<button onclick="applyOperator('Sz4')">Apply Sz4</button>

<br><br>
<!-- Measurement Buttons -->
<button id ="measureOperatorSx1">Measure Sx1</button>
    <button id ="measureOperatorSz1">Measure Sz1</button>
<button id ="measureOperatorSx2">Measure Sx2</button>
<button id ="measureOperatorSz2">Measure Sz2</button>
<button id ="measureOperatorSx3">Measure Sx3</button>
<button id ="measureOperatorSz3">Measure Sz3</button>
<button id ="measureOperatorSx4">Measure Sx4</button>
<button id ="measureOperatorSz4">Measure Sz4</button>

<br>

<button id="MeasureLX">Measure Logical Z Operator</button>
<button id="MeasureLZ">Measure Logical Z Operator</button>

<br><br>
    <div id="timer">Timer: 0</div>
<div id="currentState">
<div>
<label for="measurementResult">Measurement Result:</label>
<span id="measurementResultValue">Not measured yet</span>
</div>
    </div>

    <div id="currentState"></div>
    <svg id="canvas" width="2000" height="2000"></svg>

    <script>
document.getElementById('MeasureLX').addEventListener('click', function() {
measurement('X', [1, 2, 3]);
});

document.getElementById('MeasureLZ').addEventListener('click', function() {
measurement('Z', [1, 4, 7]);
});
```

```
document.getElementById("measureOperatorSx1").addEventListener('click', function() {
measurement('X', [1,2,4,5]);
});

document.getElementById("measureOperatorSx2").addEventListener('click', function() {
measurement('X', [3,6]);
});

document.getElementById("measureOperatorSx3").addEventListener('click', function() {
measurement('X', [4,7]);
});

document.getElementById("measureOperatorSx4").addEventListener('click', function() {
measurement('X', [5,6,8,9]);
});

document.getElementById("measureOperatorSz1").addEventListener('click', function() {
measurement('Z', [1,2]);
});

document.getElementById("measureOperatorSz2").addEventListener('click', function() {
measurement('Z', [2,3,5,6]);
});

document.getElementById("measureOperatorSz3").addEventListener('click', function() {
measurement('Z', [4,5,7,8]);
});

document.getElementById("measureOperatorSz4").addEventListener('click', function() {
measurement('Z', [8,9]);
});

let timer = 0;
        let stateIndex = 0;
        let stateObjects = [];
        let animationIntervalId;
        let timerIntervalId;
        let sumOfCoefficients = 0;
        let totalDuration;
let displayStateObjects = [
// Initialize your displayStateObjects here, for example:
// { basisVector: "111111000", coeffValue: 1 }
];
let newStateObjects = [];
let measurementResult = +1;

function measurement(operatorType, bits) {
let oldStateObjects = [];
```

```
// Apply the 0.5 * (1 + Operator) operation to each state
stateObjects.forEach((obj) => {
// First, append the original state scaled by 0.5
oldStateObjects.push({
state: obj.state,
coeffSign: obj.coeffSign,
coeffValue: obj.originalCoeff,
originalCoeff: obj.originalCoeff,
coeffFraction: `${obj.originalCoeff} / ${1}`
});
});

applyProjector(operatorType, bits, +1);

prob_plus = calculateDotProduct(oldStateObjects, stateObjects);

const randomValue = Math.random();

if (randomValue < prob_plus) {
stateObjects = normalizeStateObjects(stateObjects);
measurementResult = +1;
} else {
measurementResult = -1;
stateObjects = [...oldStateObjects];
applyProjector(operatorType, bits, -1);
stateObjects = normalizeStateObjects(stateObjects);
stateObjects = simplifyStateObjects(stateObjects);
}

// Update the measurement result on the screen
document.getElementById("measurementResultValue").textContent = measurementResult;

console.log(stateObjects);
updateDisplayedState();
}
function updateDisplayedState() {
let displayedState = "";
for(const { state, coeffSign, originalCoeff } of stateObjects) {
displayedState += (coeffSign > 0 ? "+" : "-") + `${originalCoeff} |${state}> `;
}
document.getElementById("displayState").innerText = displayedState;
}


function simplifyStateObjects(stateObjects) {
const stateMap = new Map();

stateObjects.forEach((obj) => {
const ket = `|${obj.state}>`;
```

```
const coefficient = obj.coeffSign * obj.coeffValue;

if (!isNaN(coefficient)) {
if (stateMap.has(ket)) {
stateMap.set(ket, stateMap.get(ket) + coefficient);
} else {
stateMap.set(ket, coefficient);
}
}
});

// Remove basis vectors with 0 coefficient
for (const [ket, coefficient] of stateMap) {
if (coefficient === 0) {
stateMap.delete(ket);
}
}

const simplifiedStateObjects = [];

for (const [ket, coefficient] of stateMap) {
simplifiedStateObjects.push({
state: ket.replace(/^\|(.+)\>$/, '$1'), // Extract the state from the ket
coeffSign: coefficient > 0 ? 1 : -1,
coeffValue: Math.abs(coefficient),
originalCoeff: Math.abs(coefficient).toFixed(2).toString(),
coeffFraction: `${Math.abs(coefficient)}`
});
}

return simplifiedStateObjects;
}

function normalizeStateObjects(simplifiedStateObjects) {
let sumOfSquares = 0;

// Calculate sum of squares for normalization
simplifiedStateObjects.forEach((obj) => {
const coefficient = obj.coeffSign * obj.coeffValue;
sumOfSquares += Math.pow(coefficient,2);
});

const normalizedFactor = 1 / Math.sqrt(sumOfSquares);
const normalizedStateObjects = [];

simplifiedStateObjects.forEach((obj) => {
const coefficient = obj.coeffSign * obj.coeffValue;
const normalizedCoeff = Math.abs(coefficient * normalizedFactor);
normalizedStateObjects.push({
```

```
state: obj.state,
coeffSign: coefficient > 0 ? 1 : -1,
coeffValue: normalizedCoeff,
originalCoeff: normalizedCoeff.toFixed(2).toString(),
coeffFraction: `${Math.abs(coefficient)} / ${sumOfSquares.toFixed(2)}`
});
});

return normalizedStateObjects;
}

function calculateDotProduct(originalStateObjects, newStateObjects) {
let dotProduct = 0;

// Loop through each pair of states to calculate the dot product
for (const original of originalStateObjects) {
for (const newState of newStateObjects) {
// Check if states are the same
if (original.state === newState.state) {
dotProduct += original.coeffValue*original.coeffSign * newState.coeffValue*newState.coeffSign;
}
}
}

return dotProduct;
}

function generateStateObject(terms) {
const stateMap = new Map();
let sumOfSquares = 0;

terms.forEach((term) => {
const [sign, coefficient, ket] = term.match(/([+-]?)([0-9]*\.?[0-9]+)?\|([01]+)\>/).slice(1);
const parsedCoefficient = parseFloat(coefficient || "1");
const coeffValue = (sign === '-' ? -1 : 1) * parsedCoefficient;

if (stateMap.has(ket)) {
stateMap.set(ket, stateMap.get(ket) + coeffValue);
} else {
stateMap.set(ket, coeffValue);
}
});

for (const [ket, coefficient] of stateMap) {
if (coefficient !== 0) {
sumOfSquares += coefficient * coefficient;
}
}
```

```
const normalizedFactor = 1 / Math.sqrt(sumOfSquares);
const stateObjects = [];

for (const [ket, coeffValue] of stateMap) {
if (coeffValue !== 0) {
const normalizedCoeff = Math.abs(coeffValue * normalizedFactor);
stateObjects.push({
state: ket,
coeffSign: coeffValue > 0 ? 1 : -1,
coeffValue: Math.pow(normalizedCoeff, 2),
originalCoeff: normalizedCoeff.toFixed(2).toString(),
coeffFraction: `${Math.abs(coeffValue)} / ${sumOfSquares.toFixed(2)}`
});
}
}

return stateObjects;
}

function startAnimation() {
// Clear any existing intervals or timeouts
clearInterval(timerIntervalId);
clearTimeout(animationIntervalId);
timer = 0;
stateIndex = 0;

const gridSize = document.getElementById("gridSize").value;
const [rows, cols] = gridSize.split(',').map(Number);
let quantumState = document.getElementById("quantumState").value.replace(/\s+/g, '');

totalDuration = parseFloat(document.getElementById("duration").value);

// Generate terms to be simplified and normalized
const splitStates = quantumState.split(/([+-]?[0-9]*\.?[0-9]+\|[01]+\>)/).filter(Boolean);
stateObjects = generateStateObject(splitStates);

// First, simplify the stateObjects
stateObjects = simplifyStateObjects(stateObjects);

// Then, normalize the stateObjects
stateObjects = normalizeStateObjects(stateObjects);

sumOfCoefficients = stateObjects.reduce((acc, { coeffValue }) => acc + Math.abs(coeffValue), 0);

// Initialize timer and updateState
timerIntervalId = setInterval(() => {
document.getElementById("timer").innerText = "Timer: " + timer.toFixed(2);
timer += 0.1;
}, 100);
```

```
updateState(rows, cols);

console.log(stateObjects);
updateDisplayedState();
}

        function updateState(rows, cols) {
if (stateIndex >= stateObjects.length) {
                stateIndex = 0;
                timer = 0;
            }

            const { state, coeffSign, coeffValue } = stateObjects[stateIndex];
            const timeForThisState = coeffValue * (totalDuration / sumOfCoefficients) * 1000;

            drawGrid(rows, cols, state, coeffSign);
            stateIndex++;

            animationIntervalId = setTimeout(() => {
                updateState(rows, cols);
            }, timeForThisState);
        }

        function stopAnimation() {
            clearInterval(timerIntervalId);
            clearTimeout(animationIntervalId);
            document.getElementById("timer").innerText = "Timer: stopped";
        }

function drawGrid(rows, cols, state, coeffSign) {
const svg = d3.select("#canvas");
svg.selectAll("*").remove();

let cellSize = 180;
let offsetX = 100;
let offsetY = 100;
let radius = 20;
let arrowLength = 20;
let arrowColor = coeffSign > 0 ? "#FF0000" : "#34eb46";

// Locate the coefficient information for the current state
const currState = stateObjects.find(obj => obj.state === state);
if (currState) {
// Add the fractional coefficient to the SVG canvas
svg.append("text")
.attr("x", offsetX)
.attr("y", offsetY - 100)  // Adjust the position to wherever you'd like to display the fraction
.text(`Coefficient: ${currState.coeffFraction}`)
```

```
.attr("font-family", "Times-New Roman")
.attr("font-size", "24px")
.attr("fill", "black");
}

// Draw a blue half-circle above the first square, properly aligned
svg.append("path")
.attr("d", `M ${offsetX} ${offsetY} A ${cellSize/2} ${cellSize/2} 0 0 1 ${offsetX + cellSize} ${offsetY}`
.attr("fill", "#0000FF");

// Draw a rightward-pointing blue half-circle above the quantum bits 3-6
svg.append("path")
.attr("d", `M ${offsetX + 2*cellSize} ${offsetY} A ${cellSize/2} ${cellSize/2} 0 0 1 ${offsetX + 2*cellSi
.attr("fill", "#38387F");

// Draw a downward-pointing blue half-circle above the quantum bits 8-9
svg.append("path")
.attr("d", `M ${offsetX + 2*cellSize} ${offsetY + 2*cellSize} A ${cellSize/2} ${cellSize/2} 0 0 1 ${offse
.attr("fill", "#0000FF");
// Draw a left-pointing blue half-circle above the quantum bits 4-7
svg.append("path")
.attr("d", `M ${offsetX + cellSize - 20} ${offsetY + cellSize} A ${cellSize/2} ${cellSize/2} 0 0 0 ${offs
.attr("fill", "#38387F");



// Draw lines connecting qubits horizontally and vertically
for (let r = 0; r < rows; r++) {
for (let c = 0; c < cols - 1; c++) {
let x1 = offsetX + c * cellSize;
let y1 = offsetY + r * cellSize;
let x2 = x1 + cellSize;
let y2 = y1;

svg.append("line")
.attr("x1", x1)
.attr("y1", y1)
.attr("x2", x2)
.attr("y2", y2)
.attr("stroke", "black")
.attr("stroke-width", 1);  // Change the stroke-width here
}
}

for (let c = 0; c < cols; c++) {
for (let r = 0; r < rows - 1; r++) {
let x1 = offsetX + c * cellSize;
let y1 = offsetY + r * cellSize;
let x2 = x1;
```

```
let y2 = y1 + cellSize;

svg.append("line")
.attr("x1", x1)
.attr("y1", y1)
.attr("x2", x2)
.attr("y2", y2)
.attr("stroke", "black")
.attr("stroke-width", 1);  // Change the stroke-width here
}
}

// Draw the chessboard squares
for (let r = 0; r < rows - 1; r++) {
for (let c = 0; c < cols - 1; c++) {
let color = ((r + c) % 2 === 0) ? "#38387F" : "#0000FF";
svg.append("rect")
.attr("x", offsetX + c * cellSize)
.attr("y", offsetY + r * cellSize)
.attr("width", cellSize)
.attr("height", cellSize)
.attr("fill", color);
}
}


// Draw circles, arrows, and numbering
for (let r = 0; r < rows; r++) {
for (let c = 0; c < cols; c++) {
let index = r * cols + c;
let cx = offsetX + c * cellSize;
let cy = offsetY + r * cellSize;

// Draw circles
svg.append("circle")
.attr("cx", cx)
.attr("cy", cy)
.attr("r", radius)
.attr("stroke", "black")
.attr("stroke-width", 3)
.attr("fill", "white");

// Draw arrows centrally within the circle
let arrowStartY = state.charAt(index) === '0' ? (cy + arrowLength / 2) : (cy - arrowLength / 2);
let arrowEndY = state.charAt(index) === '0' ? (cy - arrowLength / 2) : (cy + arrowLength / 2);

let arrowShaft = `M ${cx} ${arrowStartY} L ${cx} ${arrowEndY}`;
let arrowHead;
```

```
if (state.charAt(index) === '0') {
arrowHead = `M ${cx} ${arrowEndY} L ${cx - 5} ${arrowEndY + 10} L ${cx + 5} ${arrowEndY + 10} Z`;
} else {
arrowHead = `M ${cx} ${arrowEndY} L ${cx - 5} ${arrowEndY - 10} L ${cx + 5} ${arrowEndY - 10} Z`;
}

svg.append("path")
.attr("d", arrowShaft)
.attr("stroke", arrowColor)
.attr("stroke-width", 2);

svg.append("path")
.attr("d", arrowHead)
.attr("fill", arrowColor);



// Add numbering
svg.append("text")
.attr("x", cx + 15)
.attr("y", cy - 15)
.text(index + 1)
.attr("font-family", "Times-New Roman")
.attr("font-size", "24px")
.attr("fill", "red");
}
}
}

function applyOperator(type) {
            let bits;
            if (type === 'X') {
                bits = document.getElementById("Xbits").value.split(',').map(Number);
                applyX(bits);
            } else if (type === 'Z') {
                bits = document.getElementById("Zbits").value.split(',').map(Number);
                applyZ(bits);
            } else if (type === 'LogicalX') {
                applyX([1, 2, 3]);
            } else if (type === 'LogicalZ') {
                applyZ([1, 4, 7]);
            } else if (type.startsWith('S')) {
                applyStabilizer(type);
            }

            // You would need to re-calculate and re-render the quantum state
            // For demonstration, just print the updated state
            console.log(stateObjects);
updateDisplayedState();
```

```
        }

        function applyX(bits) {
            // Code to apply Pauli X operation to the specified bits
            stateObjects.forEach(obj => {
                bits.forEach(bit => {
                    let charArray = Array.from(obj.state);
                    charArray[bit - 1] = charArray[bit - 1] === '0' ? '1' : '0';
                    obj.state = charArray.join('');
                });
            });
        }

        function applyZ(bits) {
            // Code to apply Pauli Z operation to the specified bits
            stateObjects.forEach(obj => {
                let negCount = 0;
                bits.forEach(bit => {
                    if (obj.state[bit - 1] === '1') negCount++;
                });
                if (negCount % 2 === 1) obj.coeffSign *= -1;
            });
        }

        function applyStabilizer(type) {
            // Code to apply Stabilizers
            let bits;
            switch (type) {
                case 'Sx1':
                    bits = [1, 2, 4, 5];
                    applyX(bits);
                    break;
                case 'Sz1':
                    bits = [1, 2];
                    applyZ(bits);
                    break;
                case 'Sx2':
                    bits = [3, 6];
                    applyX(bits);
                    break;
case 'Sz2':
                    bits = [2, 3, 5, 6];
                    applyZ(bits);
                    break;
case 'Sx3':
                    bits = [4, 7];
                    applyX(bits);
                    break;
case 'Sz3':
```

```
                    bits = [4, 5, 7, 8];
                    applyZ(bits);
                    break;
case 'Sx4':
                    bits = [5, 6, 8, 9];
                    applyX(bits);
                    break;
case 'Sz4':
                    bits = [8, 9];
                    applyZ(bits);
                    break;
                default:
                    console.log('Unknown stabilizer type:', type);
                    break;
            }
        }



function applyProjector(operatorType, bits, projectorSign) {
// Initialize an empty array to hold the new states
let newOperatorStateObjects = [];

// Apply the 0.5 * (1 +/- Operator) operation to each state
stateObjects.forEach((obj) => {
// First, append the original state scaled by 0.5
newOperatorStateObjects.push({
state: obj.state,
coeffSign: obj.coeffSign,
coeffValue: 0.5 * obj.originalCoeff,
originalCoeff: 0.5 * obj.originalCoeff,
coeffFraction: `${0.5 * obj.originalCoeff} / ${1}`
});
});


if (operatorType === 'X'){
applyX(bits)
}
else if (operatorType === 'Z'){
applyZ(bits)
}

if (projectorSign == -1){
stateObjects.forEach((obj) => {
obj.coeffSign = -1*obj.coeffSign,
obj.coeffValue = 0.5* obj.coeffValue,
obj.originalCoeff = 0.5*obj.originalCoeff,
coeffFraction = `${obj.originalCoeff} / ${1}`
```

```
});
}
else {
stateObjects.forEach((obj) => {
obj.coeffValue = 0.5* obj.coeffValue,
obj.originalCoeff = 0.5*obj.originalCoeff,
coeffFraction = `${obj.originalCoeff} / ${1}`
});
}

// Merge new states with existing ones
stateObjects = [...stateObjects, ...newOperatorStateObjects];

// Apply only the simplification function
stateObjects = simplifyStateObjects(stateObjects);

}
    </script>
</body>
</html>
```