The Heavy-Hex Box: Allowing For Visual Debugging in Low-level Quantum Software Engineering

Samantha Norrie
University of Victoria
Victoria, Canada
[REDACTED]

ABSTRACT

With the field of quantum software engineering rapidly growing, it is crucial that those wanting to get involved have a variety of learning tools available to them. This work presents the *Heavy-Hex Box*, an exploratory learning tool designed to help facilitate the learning of and development on IBM's latest quantum computers. In addition to presenting the tool itself, this work also presents a study on the tool as well as sample exercises that can be used to encourage creative exploration with the tool.

KEYWORDS

Quantum Software Engineering, Quantum Computing Education, Visual Learning, heavy-hex lattice topology, transpilation

ACM Reference Format:
Samantha Norrie. 2024. The Heavy-Hex Box: Allowing For Visual Debugging in Low-level Quantum Software Engineering. In *Proceedings of CSC 513: Designing Creativity Support Tools (CSC 513).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

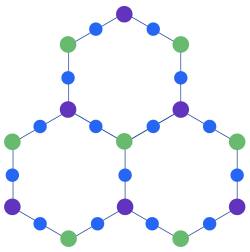

Figure 1: An abstract visualization of multiple heavy-hex lattice qubits topology formations. The blue dots denote control qubits, referred to as *C* on the HHB. The green and purple dots denote target qubits on the HHB, referred to as *T* and *t* respectively.

1 INTRODUCTION AND MOTIVATION

The field of quantum software engineering is expanding like never before. Up until a few years ago, those holding PhDs in physics were the most desired candidates for jobs in quantum. Now that quantum computers are on the cusp of being used for a variety of applications, however, candidates with software engineering and/or computer science experience are now wanted [31]. Unfortunately, limited resources are available for teaching quantum computing from a software engineering/ computer science background because most educational resources for quantum are rooted in physics.

This work aims to help contribute to filling this gap in resources by providing an exploratory learning tool designed to help teach low-level programming concepts in quantum software engineering. The tool is titled the Heavy-Hex Box (HHB). Through the HHB, users explore critical concepts such as qubit topology, transpilation, and noise. It lets users interact with and receive visual feedback from IBM's latest quantum hardware. This allows them to to practice their skills and explore new concepts using industry standard practices.

This is done by giving them a code editor where they can code quantum programs and run them in real time on a quantum computer, and an output device (referred to as the box component in this paper) that presents visual feedback received from the quantum computer. The box component prototype presented in this paper allows users to see how their quantum programs behave when heavy-hex lattice topology is used.

2 BACKGROUND

2.1 Quantum Background

Before explaining how the HHB works, a few quantum terms and concepts will be defined to give context to the tool:

2.1.1 Superconducting Qubits, Noise, and Quantum Error Correction. Qubits are the quantum equivalent to classical binary bits. Like bits, qubits can exist in the states 0 and 1, but they can also exist in states between 0 and 1. A common analogy for comparing bits and qubits is that bits are like an on/off light switch, and qubits are like a light switch with a dimmer.

Superconducting qubits are a type of qubit that are made using silicon chips and superconductors [29]. IBM's quantum chips use superconducting qubits. One disadvantage to using superconducting qubits is that they can only operate at temperatures between 10-20 millikelvin, meaning that significant work and energy needs to be put into cooling them [22]. Even a small deviation in the temperature of the system can cause erroneous results to be received. Any disruption, error, or unintended behaviour that occurs during

CSC 513, December 02, 2024, Victoria, BC

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

the run of a quantum program is called *noise*. Today's quantum era is referred to as the *Noisy Intermediate Scale Quantum* (NISQ) era because the quantum computers currently used contain a limited number of qubits and are prone to noise if one is not careful [11].

Quantum error correction is the process of preventing noise from happening. It is achieved through careful design of how qubits are placed on a quantum chip, how the qubits are used, and of how quantum programs use the machine.

2.1.2 Heavy-Hex Lattice topology. Throughout the last decade, IBM has tried several different qubit layouts on their chips to help with quantum error correction. In 2021, they decided to have the qubits on their chips organized using a heavy-hex lattice topology [25]. An abstract drawing of the topology can be seen in Figure 1. In the heavy-hex lattice topology, qubits are separated into three categories: *C*, *T*, and *t* qubits. C qubits are control qubits, and both T and t qubits are target qubits. When multi-qubit gates are applied to a set of qubits, the control qubit(s) dictate whether an operation is applied to the target qubit(s) or not. A simple example of this is the *Controlled-Not* (also referred to as *CNOT*) gate. When the qubit used as the control qubit has a value of 1, a NOT (also referred to as X) gate is applied to the target qubit. The difference between T and t qubits are the fixed frequencies they are kept at. They are kept at different frequencies that are off-resonant to prevent noise. Because the HHB is designed to support the learning of low-level programming in quantum computing and not for the learning of quantum mechanics in quantum hardware, T and t qubits behave the same in the stool.

2.1.3 Qiskit. Qiskit is IBM's quantum library [18]. It can be used to run quantum programs on a variety of quantum computers, including those not made by IBM. Qiskit is used in the HHB.

2.1.4 Quantum Gates and Transpilation. Like classical gates with bits, quantum gates change the state of the qubit(s) they are applied to. The universal gate set refers to the set of all quantum gates. While many high-level quantum libraries, such as Qiskit, allow for the universal gate set to be used when writing quantum programs, only a subset of gates are used at the hardware level. The gates allowed at the hardware level varies depending on the type and model of the quantum chip being used. The process of converting a quantum program that has been made using the universal gate set to using only the gates that are compatible with the quantum chip being used is called *transpilation*.

2.2 Quantum Education Background

IBM has created multiple resources for learning about their quantum chips. Their *Quantum Roadmap* gives a high-level overview of the different chips and quantum technology that have been created by the company as well as those that are projected to be created within the next ten years [20]. IBM also has a live dashboard that displays the current statuses of their quantum processors and describes details about their configurations [19].

While it may be mentioned, qubit topology is not often discussed in quantum computing classrooms. In Meyer et al's work, *Introductory quantum information science coursework at US institutions: Content coverage*, it was found that only 14% of introductory quantum computing classes cover qubit topology [8]. Despite this,

Figure 2: The HHB with all of its LEDs on. This is done at the beginning of each run to ensure that all LEDs are working properly.

experience in low-level quantum software engineering concepts like qubit topology are often desired in quantum computing internship applications [15]. This shows that the bridge between the education students receive in the classroom and the education they receive during an internship involving low-level concepts must be given by other sources (ex. academic research experience), or constructed by the students themselves.

3 THE HEAVY-HEX BOX: HOW TO USE

The following steps explain how to run a quantum program on the HHB:

(1) Plug the HHB hardware into your device.
(2) Navigate to *CodeEditor.py* and ensure that the port that the HHB is on is being properly connected to through *Serial* (seen in Figure 7). A comment in the code denotes where the port value should be modified if necessary.
(3) Launch the HHB code editor (seen in Figure 5) by running CodeEditor.py.
(4) Enter in the Qiskit code for the desired quantum program below the comment in the code box. The code entered must follow the following rules:
    - code above the comment must not be modified
    - all quantum gates must be appended to the *qc* variable
    - Control and target operations must only be done on control and target qubits respectively. See Figure 3 to see how each qubit index maps to an LED on the HHB

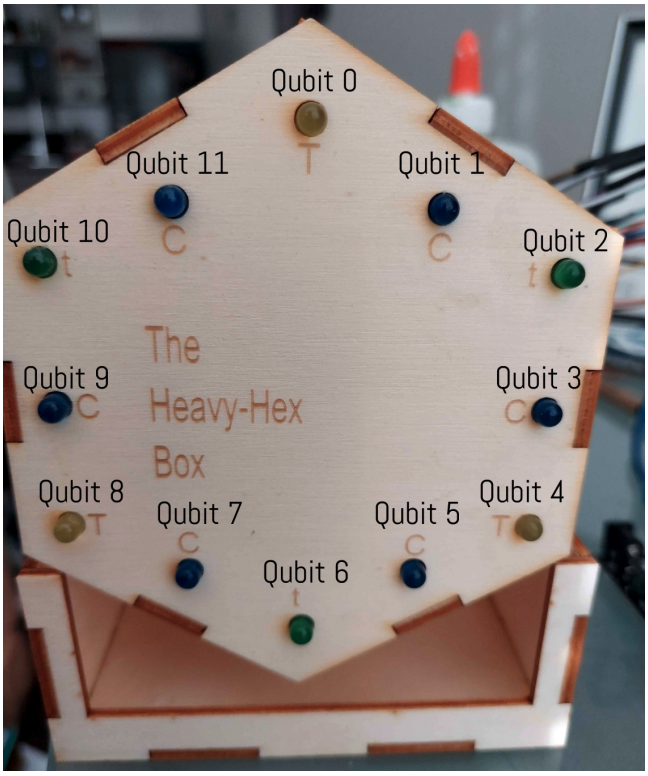

Figure 3: A diagram showing how qubit indices in the HHB's code editor map to the box component of the system. The indices were laid out in a clock-like manner to make the indexing easy for users to remember.

Alternatively, sample code can also be generated using the example dropdown at the top of the editor. Figure 5 shows the *transpilation needed + no violation* example in the editor.

(5) Check the *Enable Transpilation* checkbox if you would like transpilation to applied your quantum program. If transpilation is disabled, only the following gates can be used [16]:
- CONTROLLED-Z (CZ)
- ROTATIONAL-Z (RZ)
- SQUARE-ROOT-NOT (SX)
- NOT (X)

(6) Check the *Enable Noise* checkbox if you would like it to be possible that noise occurs while your quantum program is being run.

(7) Click the *Execute* button once you are done. All twelve LEDs of the HHB will light up to signal that your circuit is about to be run. If not all LEDs light up, stop your program and if any wires connecting the box component to your local device are loose.

(8) Observe each operation of your circuit being run on the HHB. The LED(s) representing the qubit(s) used in the operation being presented will light up. If the operation is invalid, the aforementioned LED(s) will blink until the user resubmits their code through the code editor or they turn off the device by either unplugging the hardware from their computer or close the code editor. An operation can be deemed invalid for the following reasons:
- A control qubit was used as a target qubit for the gate
- A target qubit was used as a control qubit for the gate
- An invalid gate was used (only applicable if transpilation is disabled). Figure 6 shows an example of a transpilation error being documented inside the code editor's console.

Figure 4: An operation being displayed on the HHB. In this operation, qubit 1 is being used as the control and qubit 2 is being used as the target.

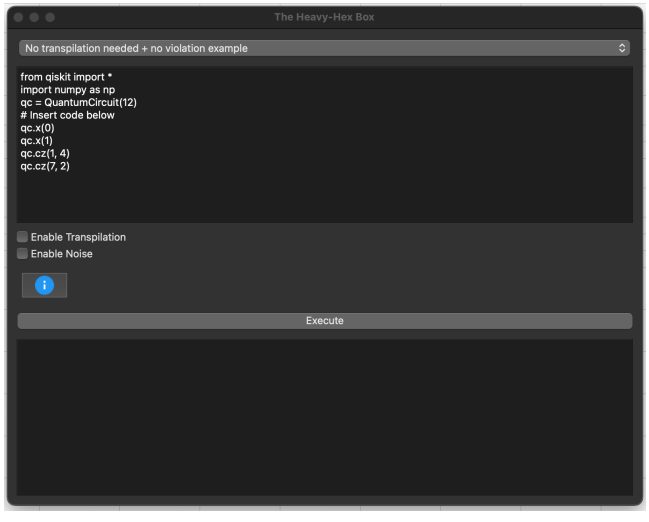

Figure 5: The Heavy-Hex Box's code editor with sample code

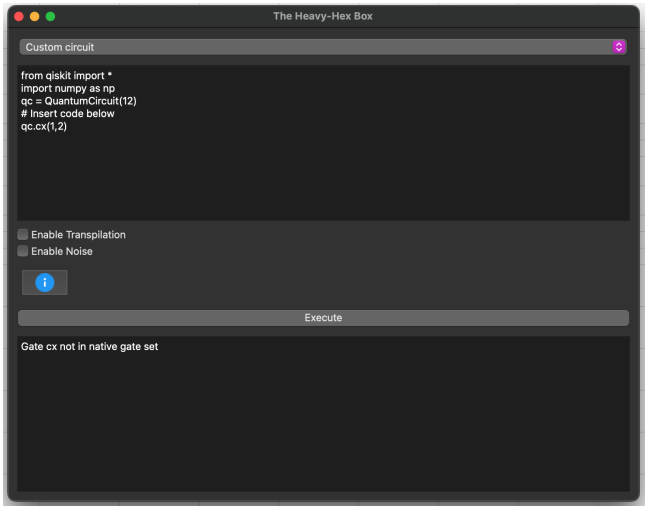

Figure 6: The Heavy-Hex Box's code editor with a transpilation error

```
def execute_code(self):
    user_code = self.code_input.toPlainText()

    output = io.StringIO()

    # CHANGE PORT HERE
    arduino = serial.Serial(port='/dev/cu.usbmodem1101', baudrate=9600, timeout=1)
```

Figure 7: The code snippet that may need to be changed upon first using the HHB. Users must update the *port* variable to the port that the HHB is on.

Operations can also be interrupted by noise, which will also cause the LEDs of the qubits involved in the operation to flash. Consult the console at the bottom of the code editor for information on why your program failed to execute. If no invalid operation or noise occurred during the run of the program, all LEDs will turn off to signal that the program has been run.

(9) Press the Execute button if you would like to run your program again.

4 RELATED WORK

The general idea for the HHB was inspired by *Tinkercad* and its slogan: 'All you need is "what if."' [7]. Like hardware projects, creating a quantum program for a real quantum computer can be costly and time intensive. The HHB's ease of use takes direct inspiration from Tinkercad's ability to lower learning and resource barriers. The workflow of the HHB was inspired by *Circuit Composer* and *QNotation* [17, 27]. Both tools let users input Qiskit code and see it iterated through in their respective visual outputs. The design of the code editor was inspired by Visual Studio Code [23].

Over the last few years, a small number of tools have been created to teach users about how qubits can be affected by their relative location to each other. *Hello Quantum* gamifies the experience of learning about how qubits interact with each other through a sleek mobile interface [32]. Users play through multiple levels where they apply gates to qubits placed in different orientations to solve a puzzle. While Hello Quantum introduces users to the idea of needing to be aware of how qubits are situated, the application does not use real-world qubit topologies. *Quantum Flytrap* is another gamified experience that lets users explore how qubits interact by dragging them and other quantum phenomena onto a virtual board [24]. Like Hello Quantum, this tool lets users learn about how relative location can affect different quantum phenomena, but it does not incorporate real world quantum topologies. In *The Qubit Game*, users build a quantum computer by controlling how qubits behave [13]. If their qubits' needs are not meant, or they are not shielded from incoming danger (noise), they may begin to negatively impact the other qubits around them. The qubits in the game roam around the screen and are never fixed to a particular location.

The quantum chip that the HHB's behaviour is based off of is the Heron chip by IBM [14]. Heron is the latest chip that has been released by the company. The hardware design of the box component of the HHB is inspired by Figure 1, which is an abstract illustration IBM released to show how heavy-hex lattices connect.

The design of the HHB's box component was inspired by the *TJBot*. The TJBot is an open-sourced project by IBM that allows users to learn about artificial intelligence. The Raspberry Pi for the TJBot sits inside of a small robot that is meant to stand on users' desks. Laser cutting and 3D printing templates are available for the TJBot.

5 PROTOTYPE DESIGN, IMPLEMENTATION, AND RATIONALE

The initial sketch of the HHB had the input portion of the system being a board where users would place blocks, representing quantum gates, together and then see the quantum circuit created be run on the box component. The input was changed to being a code editor due to the target user base of the HHB being individuals with some programming knowledge who are looking to get involved in quantum software engineering.

The box component was designed to rest next to the user's computer, ideally on their desk. A holder was designed give the box component stability. This can be seen in Figure 2. While the box and the holder components could have been 3D printed, it was decided to have both components be made out of wood through laser cutting so that it could aesthetically blend into one's desk space.

The software portion of the prototype is divided into multiple files across the two devices:

- Arduino
  - HHB_Arduino.ino
- Local device
  - QiskitUtils.py
  - Launcher.py
  - CodeEditor.py

The code for the HHB and the laser cutting template can be found on GitHub.

The Python files run on the user's local device and handle the input portion of the HHB's workflow. Launcher.py starts the application, CodeEditor.py controls the code editor that the user sees

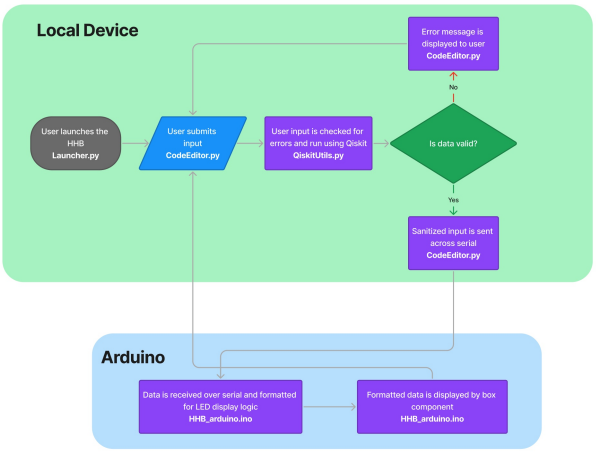

Figure 8: A diagram of the HHB's software workflow. The names of the file used during each process are bolded inside their respective process boxes.

and the sending of data over Serial, QiskitUtils.py controls the sanitization, error handling, and running of the inputted Qiskit code. The code for the Arduino handles the displaying of the data in the box component's LEDs. Figure 8 provides a visual overview of the software's workflow.

Python was chosen as the programming language for the local device portion of the system because Qiskit is a Python-based library [18]. In addition to Qiskit, the two other main libraries used for the local device portion of the system are *pyserial* and *PyQt5* [4, 30].

QiskitUtils.py contains the portion of the system that is mocked. Instead of running the quantum program on a real quantum computer, the code uses the local device to simulate the quantum program classically. This can be done when the quantum program is simple enough and is only doing computations that are solvable by a classical computer. The noise that can be experienced when running a quantum program on the HHB is fabricated and pseudo random. Currently, quantum programs with five or less gates have no chance of experiencing noise. This was done to imitate the reality that noise is more likely to happen the longer a quantum program is run.

The HHB's design rationale followed Jakob Nielsen's *10 Usability Heuristics for User Interface Design* [26]. The following list explains how the HHB addresses each heuristic:

(1) **Visibility of System Status:** The HHB informs the user that it is about to start displaying the gates from their quantum program by illuminating all LEDs. It then lights up the appropriate LEDs for each gate for the amount of time inputted by the user. The LEDs flash when an error in the quantum program has been found, indicating to the user that they should take action.
(2) **Match Between the System and the Real World:** The indices of the qubits map to the LEDs in the HHB's box component in a way that is similar to how hours map to an analog clock. The colours and positioning of the LEDs match those of the documentation put out by IBM on heavy-hex lattice quantum chips, making it easy for the user to switch between reading IBM's documentation and using the HHB.
(3) **User Control and Freedom:** The quantum program being processed by the HHB can be stopped, modified, and rerun at any time.
(4) **Consistency and Standards:** The technical wording used in the HHB (ex. heavy-hex, transpilation, noise) are consistent with what is used in the quantum computing industry.
(5) **Error Prevention:** Users are notified of code errors unrelated to those highlighted by the HHB (ex. Qiskit syntax errors) within the code editor.
(6) **Recognition Rather Than Recall:** Users are given all possible input options within the same view, and can modify them whenever.
(7) **Flexibility and Efficiency of Use:** Users can choose between using a pre-coded example or writing their own code as input.
(8) **Aesthetic and Minimalistic Design:** The HHB's code editor only displays the necessary input components on launch. Users can choose to display more information on the system by clicking on the *info* button.
(9) **Help Users Recognize, Diagnose, and Recover from Errors:** The HHB's box component indicates which qubits are causing a qubit-type or transpilation error by flashing the LEDs that are mapped to the qubits involved in the error. The LEDs will continue to flash until the user acknowledges the error in the code editor. This is done to ensure that the user has seen the error. Feedback for all other errors (ex. syntax errors) will be detailed in the code editor's output box.
(10) **Help and Documentation:** The system provides help to users through an info button, commented code (to help with serial configuration), and documentation within the system's GitHub.

6 USER PROFILE

The HHB is designed for those learning about quantum software engineering. While it does assume some background knowledge in programming and quantum computing, the application can still be used by those who do not have much of a background in either field. This is due to the system using Qiskit and the documentation given to users. Qiskit is a Python library, which helps ensure that new programmers do not experience too much cognitive load due to syntax issues. The system lets users build upon their learning by providing a variety of code examples with the code editor and ample documentation. For the HHB itself, users can use the info button, the console, and the README to help them as they navigate through the system's workflow. IBM provides several lessons and an easy-to-navigate documentation page for Qiskit. The HHB's README points users to IBM's content for Qiskit gate-related issues.

Despite being a tool that can be used by those who are beginners in the field, HHB can also benefit those who are further along in

their journey. Because the system uses industry-standard terms and practices, users can use the HHB to build upon and solidify their knowledge throughout their learning journey.

While the HHB is useful for those who are in the process of learning about quantum software engineering, it may seem trivial to those who work in the field at an intermediate or senior level. While the system may not be personally useful for these individuals, it could be used to help visualize and communicate concepts to those with less experience.

7 DESIGN FICTION ANALYSIS THROUGH NARRATIVE

To help explain the scenarios that the HHB will be used in, two user stories will be presented. The first will demonstrate what it currently like to do quantum programming, and the second will detail the story world that the HHB is situated in. After, the changes and inventions needed to jump from current day to the story world will be detailed.

7.1 Testing a Quantum Circuit in 2024

Franky is currently studying to become a quantum software engineer. While he has run many quantum programs on classical simulators, he wants to see what it is like to run a quantum program on a real quantum computer. He creates an IBM Quantum account and begins looking into the options available to him [19]. He decides to create a free account, which allow him to run quantum programs on IBM quantum computers for 10 minutes every month [19]. He sees that he can send his quantum program to multiple quantum computers using IBM's new Heron chip, and decides to try running his program on one of them.

Franky submits his quantum program. It enters a queue of jobs, and is estimated to be ready in 3 hours. He sets a reminder for himself to check the IBM portal in a few hours.

After a few hours have passed, Franky excitedly opens his finished job on the IBM portal. He notices that his program did not behave quite as he expected. He spends some time modifying his code and reading through documentation to help him find where the issue in his code is. After modifying his code, he resubmits it.

7.2 Testing a Quantum Circuit in 2033

Trafalgar is currently studying to become a quantum software engineer. He can easily run quantum programs on real quantum computers from within his code editor. Similar to his process for classical software development, Trafalgar uses multiple tools and extensions to help him his learn about development. One of the tools that he uses is the HHB. He uses the tool whenever he wants to make sure that his quantum programs run ideally on IBM's latest chips.

He repeatedly uses the HHB to help him iteratively debug his code. This allows him to quickly learn about the current quantum chip he is using.

7.3 Travelling from 2024 to 2023

The story world that the HHB exists in is based upon IBM's Quantum Road Map [20]. The road map details how quantum computers will run reliably in 2033 and will be able to be incorporated into large software engineering projects such as for drug discovery and cybersecurity.

8 DESIGN FICTION ANALYSIS THROUGH CREATIVITY SUPPORT INDEX EVALUATION

In this section, the HHB is evaluated using the Creativity Support Index (CSI) detailed in Cherry and Latulipe's paper, Quantifying the Creativity Support of Digital Tools through the Creativity Support Index [9]. This study involved three participants: the author of the HHB (referred to as *Participant A*); a software engineer studying and researching quantum computing (referred to as *Participant B*); and a technical consultant at a large consulting firm who would like to learn more about quantum computing (referred to as *Participant C*).

8.1 CSI Data Collection and Calculations

The three participants entered their results into the same Excel sheet. Originally the tool presented in Cherry and Latulipe's paper was going to be given to the participants, but it could not be downloaded [9]. The sheet given to the participants was designed to mimic Cherry and Latulipe's tool. The sheets with the raw data from the participants as well as the blank sheet that was given to them can be found on GitHub. While Cherry and Latulipe state that the collaboration portion of the CSI equation can be nullified if the tool is not designed to be collaborative, it was decided that it should be kept in for this study as collaboration is important in quantum software engineering.

$$\begin{aligned} CSI = \frac{1}{3}\big[&(Collaboration1 + Collaboration2) \cdot CollaborationCount \\ &+ (Enjoyment1 + Enjoyment2) \cdot EnjoymentCount \\ &+ (Exploration1 + Exploration2) \cdot ExplorationCount \\ &+ (Expressiveness1 + Expressiveness2) \cdot ExpressivenessCount \\ &+ (Immersion1 + Immersion2) \cdot ImmersionCount \\ &+ (ResultsWorthEffort1 + ResultsWorthEffort2) \\ &\cdot ResultsWorthEffortCount\big]. \end{aligned} \tag{1}$$

$$\begin{aligned} CSI_A = \frac{1}{3}\big[&(5+6)\cdot 1 + (8+8)\cdot 1 + (9+7)\cdot 5 + (6+4)\cdot 1 + (7+7)\cdot 3 \\ &+ (8+9)\cdot 4\big] = 75.7 \approx 76 \end{aligned} \tag{2}$$

$$\begin{aligned} CSI_B = \frac{1}{3}\big[&(7+10)\cdot 1 + (10+10)\cdot 1 + (6+8)\cdot 5 + (8+6)\cdot 2 + (10+10)\cdot 1 \\ &+ (8+7)\cdot 5\big] = 76.7 \approx 77 \end{aligned} \tag{3}$$

$$\begin{aligned} CSI_C = \frac{1}{3}\big[&(6+4)\cdot 0 + (8+9)\cdot 5 + (6+8)\cdot 2 + (7+2)\cdot 1 + (8+7)\cdot 3 \\ &+ (9+10)\cdot 4\big] = 81 \end{aligned} \tag{4}$$

Equation 1 was used to calculate each participant's CSI score. Participant A, B, and C's CSI scores can be seen in Equation 2, Equation 3, and Equation 4 respectively.

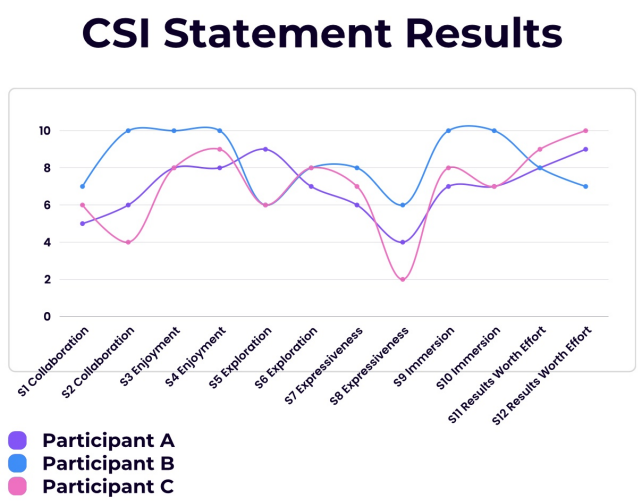

Figure 9: The participants' answers to the 12 Agreement Statements of the CSI.

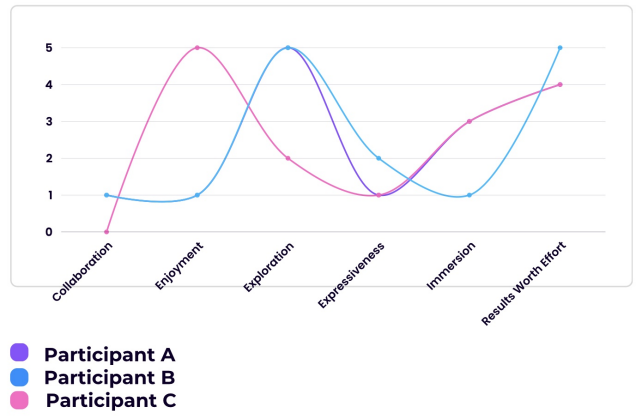

Figure 10: How participants weighed each category of the CSI. A higher number indicates higher importance.

8.2 CSI Data Analysis

Figure 9 and Figure 10 graph the points the participants gave for the 12 Agreement Statements and how heavily they weighed each category of the CSI. Despite only having a size of $N = 3$, the study was able to identify some strengths and weaknesses of the HHB:

(1) **Users enjoy the HHB.** All participants gave the enjoyment-related questions scores ≥ 8. While all participants enjoyed using the system, participants A and B gave enjoyment a weight of 1, meaning that they do not find enjoyment to be an important factor when using the tool. Participant C, however, gave enjoyment a weight of 5. It is hypothesized that this is due to Participant C being the least experienced participant.
(2) **Users' were attuned to the activity, not the system.** Statement 9, *My attention was fully tuned to the activity, and I forgot about the system or tool that I was using*, received an average score of ≈ 8.3. This indicates that, overall, the participants were not held back by impeding factors such as disorganized design, unclear instructions, or poor error handling.
(3) **Users are able to produce satisfactory results using the HHB**. Statement 12, *What I was able to produce was worth the effort I had to exert to produce it*, received an average score of ≈ 8.7. Despite being the least experienced participant, Participant C gave this statement a rating of 10. The high ratings received for this statement indicate that the system can be beneficial for those at varying points in their quantum computing learning journeys.
(4) **Users cannot effectively use the HHB as a collaborative support tool.** Statement 1, *The system or tool allowed other people to work with me easily*, and Statement 2, *It was really easy to share ideas and designs with other people inside this system or tool*, received average scores of 6 and 6.7 respectively, indicating that the HHB's collaborative capabilities ought to be improved. While the participants themselves did not rate collaboration as an important factor for them when using the tool, improving the HHB collaborative capabilities should be explored as collaboration within quantum being crucial [31].
(5) **Users are not able to be very expressive with the HHB.** Statement 8, *The system of tool allowed me to be very expressive*, received an average score of 4, which the lowest average between all 12 agreement statements. While expressiveness was not heavily weighed by any participants, it did receive weights between 1 and 2, indicating that allowing users to be more expressive in the HHB could be a feature worth exploring.
(6) **Users feel somewhat limited in their ability to explore different ideas with the HHB.** Statement 5, *It was easy for me to explore many different ideas, options, designs, or outcomes, using this system or tool* received an average score of 7. While this score is not particularly low, it does indicate that the exploratory capabilities of the HHB could be improved. Coupled with the fact that two participants gave exploration a weight of 5, it has been shown that expanding the HHB's exploratory potential ought to be prioritized in future work.

9 CREATIVE LEARNING WITH THE HEAVY-HEX BOX

This section contains multiple examples of exploratory learning exercises that can be done with the HHB.

9.1 Questions

(1) How are control qubits modified if they cannot be target qubits?
(2) How can you tell if a gate is native to the HHB?

(3) Based on the native gate set of the HHB, can you deduce which quantum chip your code is being run on?
(4) What can contribute to noise? Run multiple programs on the HHB to test your hypothesis.

9.2 Sample Answers

(1) By applying gates that do not have control/target functionality (ex. NOT-gate).
(2) If it can be used in a quantum program without transpilation.
(3) Yes, it is the Heron chip.
(4) As a quantum program becomes longer, the likelihood of noise increases.

10 CREATIVE SUPPORT TOOL QUALIFICATION AND ANALYSIS

The HHB supports the following listed stages of Sawyer's Eight Stages of the Creative Process [28].

- **Find the problem (Stage 1):** The HHB allows users to visually find problems in their quantum programs.
- **Acquire the knowledge (Stage 2).** The HHB allows users to learn exactly where the issue(s) in their program are, which supports them in having creative learning discoveries that are personally significant.
- **Gather related information (Stage 3):** The HHB gives feedback on issue(s) in its users' quantum program, allowing them to gather information on how to correct their programs and write better ones in the future.
- **Incubation (Stage 4):** The HHB allows users to repeatedly run quantum programs in rapid succession, allowing them to explore different possibilities.

The following list details how The HHB meets the requirements for being a creativity support tool (CST). The requirements have been taken from Frich et al.'s paper *Mapping the landscape of creativity support tools in HCI* [12].

(1) **The CST runs on one or more digital systems.** The HHB satisfies this requirement because it uses two digital systems that work together: the user's computer and the box component, whose output is controlled by an Arduino.
(2) **The CST encompasses one or more creativity-focussed features.** The HHB aims to be a learning tool that supports users throughout their learning journey by allowing them to have a variety of mini-c learning experiences [21]. Section 9 provides some example questions that can be used to provoke these experiences.
(3) **The CST is employed to positively influence users of varying expertise.** The HHB supports users of varying expertise by providing an easy-to-use user interface, clear instructions and documentation on the system, and examples for beginners to use to get started.
(4) **The CST supports one or more distinct phases of the creative process.** As shown in the previous list, The HHB supports four stages of Sawyer's Eight Stages of the Creative Process.

11 TANGIBLE USER INTERFACE ANALYSIS

Due to it having both a physical and digital components, it is possible that the HHB qualifies as a tangible user interface (TUI). The four characteristics of TUIs defined by Ullmer and Ishii in their paper, *Emerging frameworks for tangible user interfaces*, are used in the following list to analyze the HHB's TUI potential:

(1) **Physical representations are computationally coupled to underlying digital information.** As seen in Figure 3, the HHB satisfies this requirement as the qubits that are digitally interacted with through programming inside the code editor all map to an LED on the box component.
(2) **Physical representations embody mechanisms for interactive control.** The HHB does not satisfy this requirement as the box component does not allow for interactive control.
(3) **Physical representations are perceptually coupled to actively mediated digital representations.** The HHB satisfies this requirement as the code editor provides dynamic information on the state of the system.
(4) **Physical state of tangibles embodies key aspects of the digital state of a system.** The HHB satisfies this requirement because the topology of the qubits is clearly represented in the box component.

While the HHB does not satisfy all the requirements listed, it is important to note that due to the vastness of the field and how different communities define certain terms, analyzing the HHB under certain lenses may lead some to believe that it is a TUI. For example, in their paper, Ullmer and Ishii note that physical representations may have varying degrees of freedom in their movement. They explain that some physical representations may be able to be moved freely, while others may be tightly constrained. If one defines the box component as a physical representation that is tightly constrained but still gives users interactive control (ex. by giving them information that allows them to make more informed decisions when inputting future quantum programs into the system), then the HHB can be said to be a TUI.

12 LIMITATIONS AND FUTURE WORK

One key limitation found through the study detailed in section 8.2 was that the HHB does not support collaboration. While the tool itself is designed for solo use, features ought to be completed as future work to help facilitate the sharing of results.

The addition of file import/export capabilities and wireless communication between devices will help improve the system's ability to be used collaboratively. Allowing users to import and export Qiskit code from the code editor will allow them to share ideas and test code together. Currently, the HHB must stay connected to the user's local device because serial communication is used to send data between the local device and the Arduino. By adding wireless communication, users will be able to pass around their box components and will be able to send quantum programs to other HHB systems.

Wireless communication will be able to be achieved by adding either a Bluetooth or WiFi micro-controller to the Arduino [1, 2]. The code for supporting these micro-controllers would then have to be added to *HHB_arduino.ino*, and the *Serial* object code would

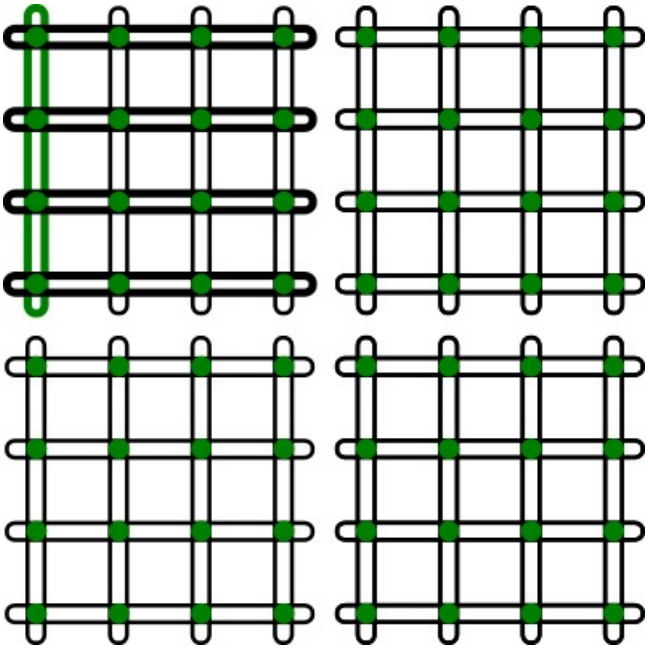

Figure 11: An abstract drawing of D-Wave's Chimera topology [10]. In the Chimera topology, qubits are represented as horizontal and vertical loops (drawn in black in this image), with the intersections (highlighted in green) being where the qubits intersect (also known as couple). The coupling of qubits allows for quantum behaviour to occur.

need to be removed to prevent clutter in the file. The *Comm.py* file would also need to be modified. Libraries such as *Pybluez* or *pyserial* can be used for Bluetooth communication, and libraries such as *websockets* or *requests* can be used for WiFi communication [3–6].

A second limitation of the HHB is that the system is made only for learning about IBM's quantum chips. It is hypothesized that this may be one of the reasons for points 5 and 6 in section 8.2. To ensure that users are able to be supported in becoming flexible quantum software engineers, quantum chips made by other quantum companies should be made into a tool like the HHB. For example, D-Wave's Chimera topology aligns its qubits in a matrix-like fashion, with the rows and columns of the matrix having effects on how the qubits should be used. An abstract example of this can be seen in Figure 11.

A third limitation of the HHB is its ability to only support up to twelve qubits/one heavy-hex lattice. Like with the second limitation, it is believed that this limitation contributed to points 5 and 6 in section 8.2. As seen in Figure 1, the quantum chips using the heavy-hex lattice topology use multiple hexagons. Future iterations of the HHB should explore allowing multiple box components to be joined together. This could be accomplished by adding magnets to the edges of the box. Having wireless communication implemented before or concurrently with this feature is necessary as users should be able to arrange and append several boxes together.

REFERENCES

[1] [n. d.]. Robojax HC-05 Bluetooth Serial Pass-Through Module Wireless Serial Communication with Button for Arduino. https://www.amazon.ca/Robojax-Bluetooth-Pass-Through-Wireless-Communication

[2] [n. d.]. WiFi Module - ESP8266 (4MB Flash). https://www.sparkfun.com/products/17146

[3] 2019. PyBluez 0.23. https://pypi.org/project/PyBluez/

[4] 2020. Welcome to PySerial's Documentation. https://pyserial.readthedocs.io/en/latest/

[5] 2024. requests 2.32.3. https://pypi.org/project/requests/

[6] 2024. websockets 14.1. https://pypi.org/project/websockets/

[7] Autodesk. [n. d.]. All you need is 'what if'. https://www.tinkercad.com/

[8] Josephine C. Meyer, Gina Passante, Steven J. Pollock, and Bethany R. Wilcox. 2023. Introductory quantum information science coursework at US institutions: Content coverage. *ArXiv* (2023). arXiv:ArXiv/physics.ed-ph/2308.12929 [Physics Education]

[9] Erin Cherry and Celine Latulipe. 2014. Quantifying the Creativity Support of Digital Tools through the Creativity Support Index. *ACM Transactions on Computer-Human Interaction* (2014).

[10] D-Wave. 2024. D-Wave QPU Architecture: Topologies. (2024). https://docs.dwavesys.com/docs/latest/c_gs_4.html#

[11] James Dargan. 2023. What Is NISQ Quantum Computing? (2023). https://thequantuminsider.com/2023/03/13/what-is-nisq-quantum-computing/

[12] Jonas Frich, Lindsay MacDonald Vermeulen, Christian Remy, Michael Mose Biskjaer, and Peter Dalsgaard. 2019. Mapping the landscape of creativity support tools in HCI. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (2019).

[13] Google Quantum AI. 2022. The Qubit Game. https://quantumai.google/education/thequbitgame

[14] IBM. 2023. IBM Debuts Next-Generation Quantum Processor IBM Quantum System Two, Extends Roadmap to Advance Era of Quantum Utility. (2023). https://newsroom.ibm.com/2023-12-04-IBM-Debuts-Next-Generation-Quantum-Processor-IBM-Quantum-System-Two,-Extends-Roadmap-to-Advance-Era-of-Quantum-Utility

[15] IBM. 2024. IBM Careers. https://www.ibm.com/careers

[16] IBM. 2024. Native Gates and Operations. (2024). https://docs.quantum.ibm.com/guides/native-gates

[17] IBM Quantum. 2023. *Composer*. quantum-computing.ibm.com/composer

[18] IBM Quantum. 2024. Qiskit. qiskit.org

[19] IBM Quantum. 2024. Quantum Processing Units. https://quantum.ibm.com/services/resources

[20] IBM Quantum Research. 2024. Quantum Roadmap. https://www.ibm.com/roadmaps/quantum/

[21] Kaufman J.C. and Beghetto R.A. 2009. Beyond Big and Little: The Four C Model of Creativity. *Review of General Psychology* (2009).

[22] Michael Martin, Caroline Hughes, Gilbert Moreno, Eric Jones, David Sickinger, Sreekant Narumanchi, and Ray Grout. 2021. Designing Energy-Efficient Quantum Computers Through Prediction and Reduction of Cooling Requirements for Cryogenic Electronics. *NREL* (2021).

[23] Microsoft. [n. d.]. Visual Studio Code. https://code.visualstudio.com

[24] Piotr Migdał, Klementyna Jankiewicz, Paweł Grabarz, Chiara Decaroli, and Philippe Cochin. 2022. Visualizing quantum mechanics in an interactive simulation – Virtual Lab by Quantum Flytrap. *Optical Engineering* (2022).

[25] Paul Nation, Hanhee Paik, Andrew Cross, and Zaira Nazario. 2021. *The IBM Quantum heavy hex lattice*. Technical Report. IBM Quantum Research.

[26] Jakob Nielsen. 1994. 10 Usability Heuristics for User Interface Design. (1994). https://www.nngroup.com/articles/ten-usability-heuristics/

[27] Samantha Norrie, Anthony Estey, Hausi Müller, and Ulrike Stege. 2024. QNotation: A Visual Browser-Based Notation Translator for Learning Quantum Computing. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*.

[28] R.K. Sawyer. 2012. Explaining creativity: The science of human innovation. (2012).

[29] Josh Schneider and Ian Smiley. 2024. What is Quantum Computing? www.ibm.com/topics/quantum-computing

[30] Phil Thompson. [n. d.]. PyQt5 5.15.11. https://pypi.org/project/PyQt5/

[31] Louise Turner and Yoan Mantha. 2024. The Canadian Ecosystem Report 2024. (2024). https://www.qai.ca/2024-quantum-ecosystem-report

[32] James Wootton. 2018. Hello Quantum: Taking your first steps into quantum computation. *Medium* (2018).