Cohort project: The Quantum Game

Solomons, Naomi R.
naomi.solomons@bristol.ac.uk

Jones, Benjamin
benjamin.jones@bristol.ac.uk

Belsley, Alex
alex.belsley@bristol.ac.uk

Palmer, Quinn
quinn.palmer@bristol.ac.uk

Bell, Tom
tj.bell@bristol.ac.uk

Currie, Sebastian
sebastian.currie@bristol.ac.uk

Stafford, Matt
matthew.stafford@bristol.ac.uk

Clark, Marcus
mj.clark@bristol.ac.uk

Fasoulakis, Anastasios
ze19163@bristol.ac.uk

A. Kulmiya, Sahra
cd19170@bristol.ac.uk

Mister, Sam
sm15883@bristol.ac.uk

August 2020

Abstract

In this report we present 3 quantum games: Zedex, Squamble and (QGame). We provide a detailed breakdown on all three games, including discussions on background theory, game play and code structure. To determine the success of these games they are evaluated against a set of criteria we set out at the start of the project. Zedex and (QGame) both tackle optimisation tasks in quantum computing while Squamble looks at quantum games as an educational tool. The games and all source code is available online (Give link to cohort 6 github?).

Contents

.

1   Introduction and background

Before the cultural explosion of video games, early games played on a classical computer were generally confined to research labs and institutions. These initial projects were not made public and served a purely academic and experimental purpose [1]. The earliest known public demonstration of a computer game was in 1950 called Bertie the brain at the Canadian national exhibition [2]. The game was Tic-tac-toe and served to give the public an interactive experience with these new machines. Once computers became available to the general public, computer games grew into an industry now worth approximately US$120 billion [3].

The advent of quantum computing has naturally led to the question of whether it is possible to use the new technology for gaming. Since 2017, quantum computing devices accessible through the cloud such as the IBMQ experience have allowed developers to make games that utilise real world machines. An extensive review of quantum games developed 2017-2019 is given in [4].

Quantum games can be categorised into two broad classes; games that either run on or otherwise use a real quantum computing device to operate, and games which are classical but incorporate quantum mechanical ideas (usually for educational purposes). The first quantum game that utilises a real machine, called Cat-Box-Scissors (a spin on Rock-Paper-Scissors) was developed by Wootton in 2017, with the randomness coming from the probabilistic nature of measurement. In 2018 Wootton developed a new game, this time with the utility of being able to benchmark quantum computers, by designing a puzzle that represented the specifications of the particular hardware it used. The game was called quantum awesomeness [5, 6]. On the other hand, games such as quantum tic-tac-toe introduce the user to the principles of superposition and entanglement, and these are used to give an interesting twist to a well known game [7]. 2019 saw the first quantum game jam - which challenged participants to develop games incorporating quantum physics [8] - and Quantum Chess [9].

Inspired by these efforts, this work presents three new quantum games. One goal of the games was to provide useful output for tasks in quantum computing, and Section 2 introduces several of these problems. Section 3 describes the systems requirements that were used to motivate and then evaluate the games. The games themselves

are described in Sections 4, 5 and 6. Appendix Section A gives an overview of the quantum hardware and software currently available.

2 Optimisation tasks

A major goal of developing a quantum game was to provide a technique for solving problems tailored to specific hardware, both as a way of providing a useful output to the game and to make use of existing quantum computers. One way of doing this is to design games based around optimisation problems, several of which are detailed in this section.

2.1 The Qubit Routing Problem

Most quantum algorithms are described via the framework of *quantum circuits*, which allow for abstraction away from the underlying hardware and provide a conceptually useful diagrammatic representation [10]. Typically, the input to a quantum circuit is taken to be $|0\rangle^{\otimes n}$, where all qubits are initialised to the computational basis state $|0\rangle$. The computation proceeds with local unitary gates, measurements, and classical feedforward before all or some of the qubits are measured in the computational basis, for example yielding a bitstring of length n. Practical considerations make it desirable to only allow one and two qubit gates: interacting three qubits simultaneously is considered to be a difficult task on any platform, not to mention the increased complexity for error-correcting multi-qubit gates. Fortunately, it suffices to consider two-qubit gates, and moreover one can generate a universal gate set (approximating any unitary to within small error) with only a single two-qubit gate (for example, the controlled-NOT) and arbitrary one-qubit gates.

When designing abstract quantum circuits, it is implicitly assumed that it would be possible to apply a two-qubit gate between any pair of qubits. In reality, on current platforms the connectivity of the qubits is limited; a two-qubit gate can only be applied on certain pairs of qubits. Figure 2 shows the connectivity of current IBMQ devices. Abstract quantum circuits can still be run on such hardware, but the use of SWAP gates is required, in order to allow non-native two-qubit gates to be performed (as shown in Figure 1). This comes at a cost, as it desirable to minimise the number of gates (and in particular, two-qubit gates) in order to minimise the errors that may occur. This problem is often referred to as the *qubit allocation problem* or *qubit routing problem* and is discussed extensively in [11, 12].

(a) Original Circuit. (b) Architecture. (c) Modified Circuit.
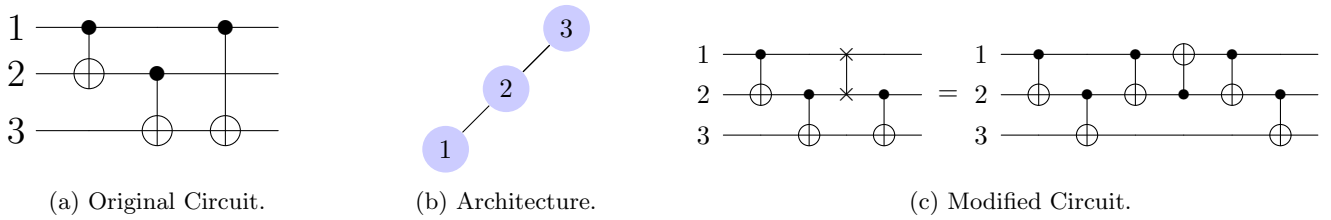
Figure 1: A example of the qubit routing problem. An initial circuit is to be implemented on an architecture, but there exists a non-native two qubit gate, in this case the final CNOT between qubits 1 and 3. A swap must be applied in order to be able to perform this gate, at the expense of three additional CNOT gates, as shown in the modified circuit.

> ## Problem Statement: Qubit Routing
>
> **Input:** A quantum circuit, and a quantum architecture. The circuit could be in the form of a list of gates, and the architecture could be represented by a graph.
>
> **Output:** A quantum circuit that can be run on the architecture natively, that implements the same unitary as the original circuit.
>
> Such that the number of additional swap gates required is minimised.

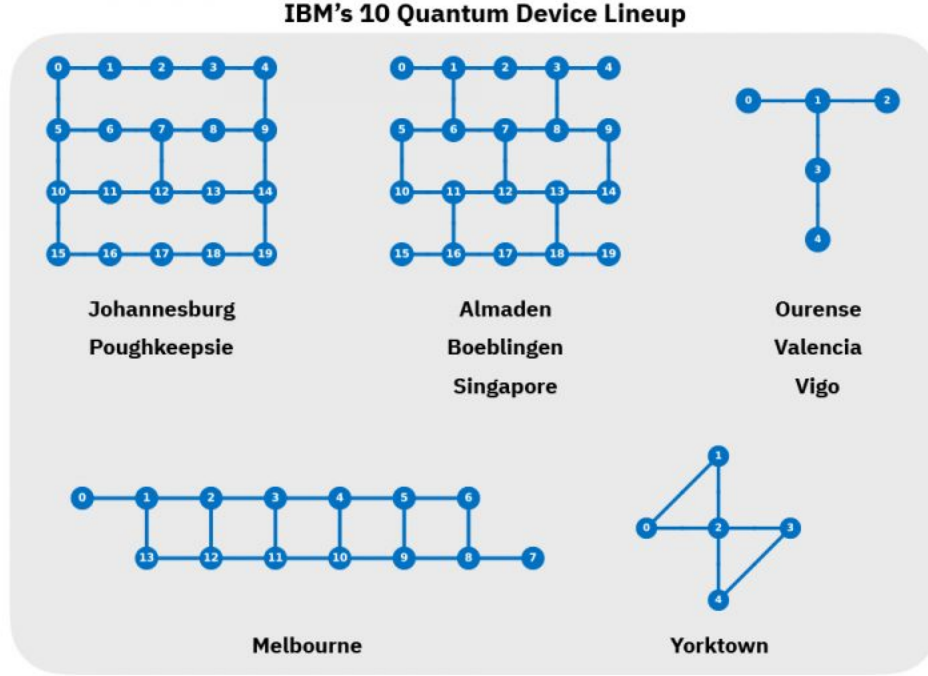Figure 2: Connectivity of current IBMQ devices. Accessed on 11-08-2020: `https://www.ibm.com/blogs/research/2019/09/quantum-computation-center/`.

Current Approaches

Consider a decision problem that is simpler than the general qubit routing problem: given a circuit and architecture, is there a mapping from logical to physical qubits such that the circuit can be directly implemented on the architecture, with no additional swaps? Note that for n qubits, there are $n!$ possible permutations, and it is known that this problem is in the complexity class **NP-Complete** [12]. Thus we do not expect this problem to be tractable as the number of qubits grows: for example [12] proposes an algorithm to solve the qubit routing problem exactly which for both running time and space usage is super-exponential in the number of qubits (and linear in the number of gates). Thus it is expected that heuristic methods must be used in practice. Recent approaches are discussed and implemented within the t|ket⟩ package [11].

Qiskit currently take the following approach. Within Qiskit all issues regarding the conversion of abstract quantum circuits into their physical implementations is handled by the transpiler and pass manager. The transpiler consists of collection of circuit transformations called 'passes'. These passes include breaking down an abstract circuit into some desired set of basic gates that are available to the chosen device. The complete transformation of abstract circuit to physical implementation will generally take multiple different passes and this is handled by the pass manager. The pass manager allows for these different processes to communicate and be chained together, and this modular design gives great control over the specific transpiling process desired. In the context of qubit routing it has a number of transpiler passes that will insert swap gates for the circuit to map onto the connectivity architecture. A few of these are *BasicSwap, LookaheadSwap,* and *StochasticSwap*. In Figure 3 we have an initial quantum circuit and the resulting transformed circuits after each of these qubit routing routines. It can be seen in the circuit diagrams that the BasicSwap routine is the worst as it uses many swaps to map the qubits onto the connectivity. The Lookahead and Stochastic swap routines fare much better but their performance can vary greatly depending on the circuit being routed.

2.2 Other optimisation problems

Qubit routing is one aspect of circuit optimisation and compilation. Something else that needs to be considered is qubit allocation. In IBM machines the fidelity for single and two qubit gates can vary depending on which qubits they are performed. For this reason, this Qiskit transpiler comes with passes for allocating qubits in the circuit to
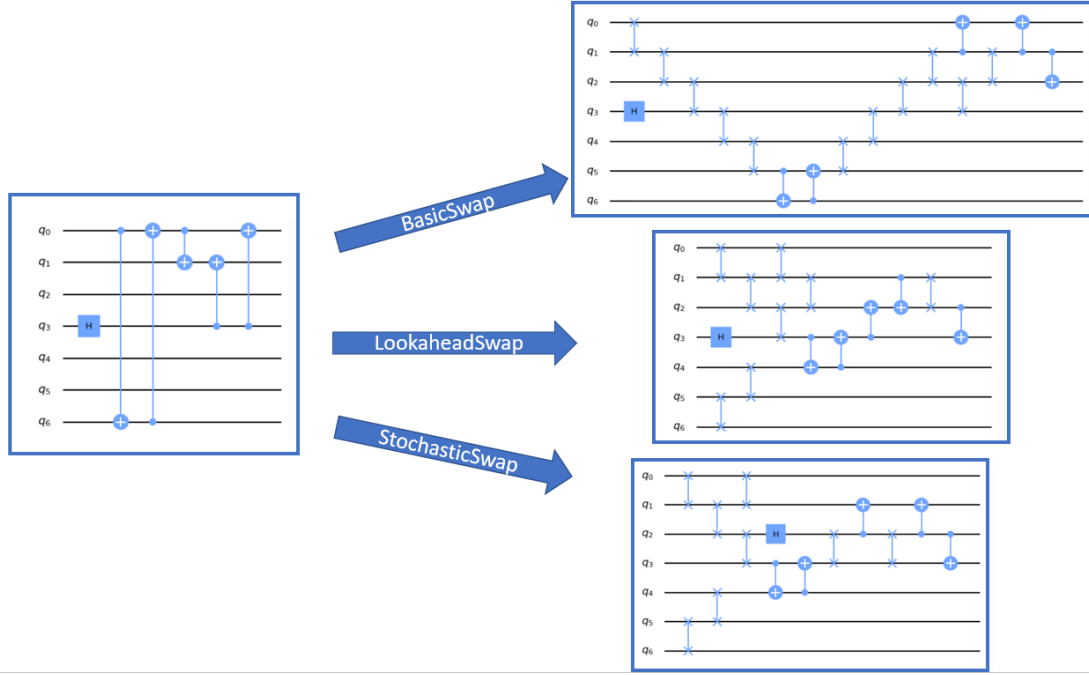
Figure 3: Starting from some initial circuit on the left, we have the resulting circuit transformation after each of the three qubit routing algorithms BasicSwap, LookaheadSwap and StochasticSwap are applied.

ones in the physical architecture. A few of these passes are *TrivialLayout, DenseLayout* and *NoiseAdaptiveLayout*. In a similar manner to the three qubit routing routines, each of these qubit allocation routines differ in sophistication and performance. Some, such as the *NoiseAdaptiveLayout*, take into account different qubit fidelities when allocating qubits. This optimisation procedure is incorporated into a game mechanic for some of the games we will present here.

While much of the performance of a given circuit will depend on qubit allocation and routing, we can imagine optimisation procedures that happen on the circuit level that have an awareness of the underlying architecture before qubit allocation and routing are performed. More specifically there may be ways to rewrite a circuit at a high level of abstraction so that it performs the same desired task but is better suited for the target architecture from the outset. Optimised circuits like this may make it easier for the qubit allocation and routing routines to achieve better performance from the circuit. An example of this kind of optimisation is present in one of our games and demonstrated in Section 4.4. The example considers a GHZ state generator circuit and some particular qubit connectivity structure. It is demonstrated that the circuit can be transformed into a variant that can map perfectly onto the connectivity required by the architecture, hence no routing is required. The circuit performs the same task of generating a 5 qubit GHZ state and this optimisation would not be achievable with qubit allocation alone. This specific optimisation however requires that the input to the circuit is always the all zero state, and so this technique would be limited to circuits where this is the case, such as in state generators.

3 Aims

We focused on a particular set of system requirements that we felt were necessary for a successful quantum game.

Educational: an aim of many of the games discussed in Section 1 was to teach concepts in quantum mechanics, such as superposition or entanglement. Games are a valuable resource for outreach work, in particular if they are well pitched to a particular demographic (which could mean spending more time on a user interface that's more easily accessible for younger ages). Many games already exist that do this effectively, however it would still be useful to have more resources to illustrate more complex ideas, such as the basic model of a quantum circuit.

Useful: one of the objectives of the project was to develop a game that provides data or insights that contribute towards ongoing research, most likely through users generating data that can be accessed by interested parties, who could potentially design their own 'levels' to be used as inputs to the game. The data could be useful if the users are tasked with solving or optimising a problem that is not efficiently solved algorithmically, particularly if commonly

used methods provide an insight into new approaches to solve the problem, or if they provide the opportunity to consider a large number of different examples to find a counter-example to disprove a theorem.

This would depend on how data from the levels can be collected, and how easily levels are generated, as well as the number of users and how often they play the game. However, it is worthwhile considering as a framework for game design even if it is not able to be implemented with our current resources.

Enjoyable: a central consideration (and typically the most important factor in traditional game design) is how enjoyable the game is to play. In this case, this could be a deciding factor not only in the popularity of the game but how much data is generated by users.

Although this is a subjective property and difficult to measure, we felt that there were some important aspects to achieve first. The game should be intuitive and easy to learn (this is important to the educational aspect of the games as well), and preferably they should have a good graphical interface. We also had to consider the response time of the interface so that the user had immediate feedback on the results of their moves, which conflicted with our end goal of running on an existing quantum computer, as discussed below. Finally, it means - particularly for optimisation-based games - having a scoring system which is clear to the user.

Accessible: we aimed to ensure the game was hosted online somewhere, or available as a phone app. It would be useful if the game is well publicised. The instructions provided with the game should be clear and intuitive (either for someone with no coding background or adjusted to suit the audience).

Hardware: an initial goal for the project was to develop a game that could feasibly be played on an actual quantum computer - either one currently existing, or a potential future model. The current state of available hardware/software for quantum computers is summarised in Section A.

Based on the idea that immediate feedback as a response to user action was necessary to make the game enjoyable and easily playable, we decided that the games should incorporate data produced by real quantum computers and generate results based on these. This also contributes to the possibility of making the games useful in producing data that benchmarks, optimises, or otherwise analyses the existing circuits and hardware, in particular by applying them to the optimisation problems discussed in Section 2.

4   Zedex

Zedex is a quantum game that utilises the ZX calculus as a means for players to rewrite and reconfigure quantum circuits. A circuit can be represented as a ZX graph object containing nodes (called spiders) and connecting edges. It has been shown that the ZX calculus is universal and complete for pure qubit quantum information, therefore a ZX object is a valid representation of a quantum circuit [13]. The ZX calculus comes equipped with a set of rewrite rules which allows for conversions between equivalent graphs, changing the makeup of the circuit's gates while still being equivalent to the overall unitary operation. Zedex utilises a subset of these rewrite rules to allow the user to manipulate the ZX object with the aim of simplifying a quantum circuit and improving its mapping to an underlying architecture. The ZX rules can be performed on the graph using intuitive clicks and movements of the mouse and the spider nodes can be dragged around as the player wishes. Users are then scored according to how well they map the circuit to the architecture and, where possible, are presented with an updated circuit to view the result of their manipulations.

Zedex is available to play by accessing [INSERT LINK], documentation for the game and tutorials can be found at [INSERT LINK] and the source code is located at [INSERT LINK].

4.1   The ZX Calculus

The backbone of Zedex is the ZX-calculus. The ZX-calculus is a graphical representation of linear maps between qubit states (such as the unitary associated with a quantum computing circuit) [14]. ZX diagrams are universal and are composed of connected spiders, which come in two different flavours depending on the basis they act upon. Z spiders are represented by green nodes and are expressed in the computational basis while red X spiders are expressed in the conjugate basis [13]. The nodes in ZX calculus also contain a phase term. A Z/X single-qubit spider with zero phase corresponds to the identity map, non-zero phases correspond to arbitrary Z/X rotation operators. Figure 4 contains several common ZX objects and their corresponding form as a linear map. Two graphs represented by the ZX calculus are equivalent if one can be converted into the other via the following set of rewrite rules [15]:

Figure 4: ZX calculus generators and the corresponding linear maps. (a) $|+\rangle$ and $|-\rangle$ input states for $\alpha = 0$ and $\alpha = \pi$ respectively; (b) Input computational basis states $|0\rangle$ and $|1\rangle$ for $\alpha = 0$ and $\alpha = \pi$ respectively; (c) Z-axis rotation by an angle α; (d) X-axis rotation by an angle α and (e) Hadamard gate. In our game, the Hadamard gate is represented by a blue line.

- Two rules to **fuse** spiders of the same colour, one with a phase α and another with a phase β, resulting in a single spider with a phase $\alpha + \beta$ modulo 2π as shown in Figure 5 (a) and (b).

- A rule to **commute** spiders with different colours. For example, it is possible to exchange a green node with a phase α with a red one with a π phase by flipping the sign of α as illustrated in Figure 5 (c).

- A rule to **change the colour** of a spider, allowing us to convert between green and red spiders. This requires the use of a Hadamard gate, which is graphically denoted by a yellow square as depicted in Figure 5 (d).

- A node with two edges and zero phase is equivalent to the **identity** and can thus be removed as shown in Figure 5 (e).

- The **bialgebra rule**, involving commutation of the multiplication and co-multiplication, allows us to replace the pattern of Z and X spiders in Figure 5 (f) by a pattern involving a single Z and X spider.

- We can decompose a single-qubit operator in terms of three rotations around the Bloch sphere - a rule commonly referred to as **Euler decomposition** - as depicted in Figure 5 (g).
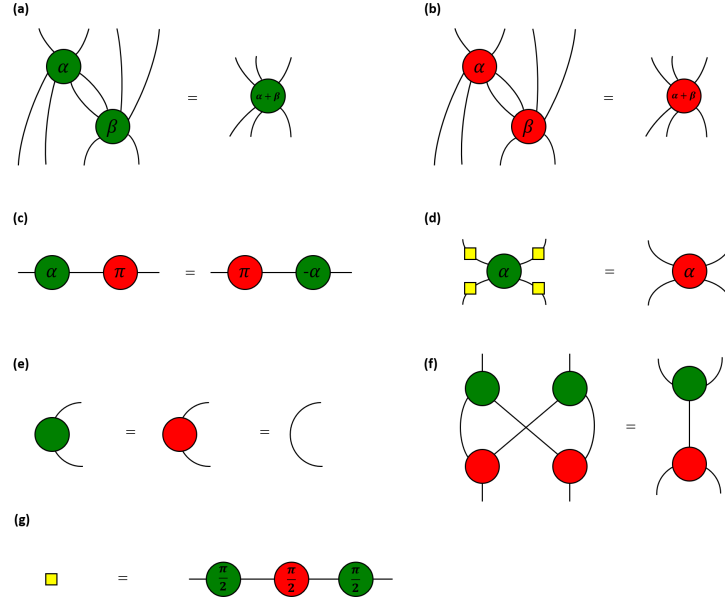

Figure 5: ZX calculus rules. (a) Z-spider fusion, (b) X-spider fusion, (c) π commutation, (d) colour change, (e) identity, (f) bialgebra and (g) Euler decomposition. The Z and X spiders are represented by green and red nodes while the Hadamard gate is symbolized by a yellow square.

4.2 ZX as a compilation optimiser

To run a quantum algorithm on underlying physical hardware the circuit must be compiled to machine level code. The compilation process involves several intermediate steps to go from an abstract circuit to physical qubit manipulations called 'pulses'. The first compilation stage is to convert the abstract algorithm to an intermediate representation (IR) of the circuit. The circuit is decomposed into a universal native gate set and undergoes rounds of hardware independent circuit optimisation with the intention to minimise overall circuit depth, two qubit and expensive gate counts. Two qubit gates are challenging to implement as they require physical connectivity between qubits, leading to increased gate error rates. If the logical gates do not match up with the physical architecture the qubits must be swapped, an example of the qubit routing problem discussed in Section 2.1.

A long term goal of circuit optimisation is to minimise the use of non fault-tolerant gates for a given algorithm. Once scaled up, quantum computers will require error correction to protect against noise on the physical qubits by redundantly encoding data into logical qubits. Logical gates from the algorithm will consist of a collection of underlying physical gates. A logical gate is said to be transversal if it does not propagate errors to multiple physical qubits when implemented. However, no code contains an exact universal, transversal logical gateset - at least one of the gates will be prone to error [16]. In the {Clifford + T} gateset using the surface code the T gate is not transversal and requires significantly more resources to implement in a fault-tolerant manner. Therefore reducing the number of these expensive operations is beneficial when attempting to map a quantum algorithm to the hardware.

The ZX calculus has been shown to provide significant gains in compilation optimisation strategies for reducing two qubit gate counts and overall T counts [17, 18]. The high level optimisation process runs as follows. First, a circuit representation is translated into its equivalent ZX graph. The circuit is first broken down into generalised Z and X rotations, which are then represented by connected ZX spiders. Rewrite rules can now be applied to the ZX graph to simplify the circuit. The ZX graph can now be reduced by applying the rewrite rules discussed above. The reduced ZX graph is then extracted back to a simplified, logically equivalent quantum circuit.

While ZX graphs are equivalent under the rewrite rules, the graph object may no longer directly represent a quantum circuit after processing. All quantum circuits can be expressed as ZX graphs but the converse is not true, at first glance it is unclear if an equivalent circuit can be extracted from the reduced graph. It has been shown

that when two circuit ZX diagrams are equivalent under the rewrite rules the corresponding circuits are logically equivalent [19]. Once the graph has been fully reduced it has to then be post-processed into a form which can be mapped to a circuit. Extraction algorithms, starting from a specific 'semi-normal form' of the ZX graph have been developed, but extraction of a general simplified graph remains a hard problem.

The optimisation strategies discussed above have been implemented in an open source python package called PyZX. This is designed to systematically optimise quantum circuits through ZX graph manipulation. PyZX contains functionality to import quantum circuits, perform pre-defined optimisation routines based upon combinations of ZX rules and re-extract the optimised object back to a circuit. Several optimisation procedures have been developed which simplify Clifford operations and reduce two qubit and T gate counts. While PyZX has demonstrated success in two qubit and T-gate optimisation, the optimisation routines do not account for underlying architectures.

4.3 The game

The objective of the game is to optimise a given circuit using ZX rewrite rules to minimise overall gate counts, and improve the mapping of two-qubit gates to a physical architecture. Users are rewarded for ensuring that two-qubit gates are on the highest fidelity connections available. Figure 6 shows the user interface for a Zedex level. A user is provided with a ZX graph representation of a circuit, in this case a five qubit GHZ generation algorithm, which they can then manipulate. Original and, where possible, optimised circuits are also displayed to the user to provide a visual representation of their actions and aid understanding. A scoring mechanism is also provided to the user for in game feedback.

Along with the ZX graph, the underlying connectivity of the physical qubits is displayed to the player graphically. The black nodes on either side of the graph represent input and output qubits. The connectivity structure is shown on the leftmost qubits, where lines between qubits indicates that the qubits are connected in the physical device. The colouring of the connectivity edges represents the relative fidelity of these controlled NOTs, where a brighter blue colour indicates a higher fidelity. This visualisation gives the player some insight into some properties of the underlying device and its purpose is to help guide the player in what moves they perform.
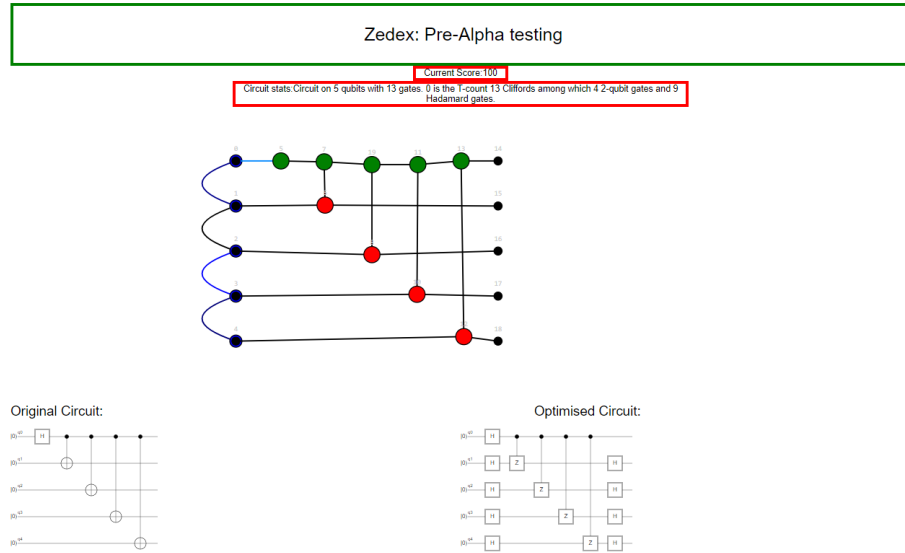
Figure 6: An example Zedex level for a 5 qubit GHZ state generation circuit. The main graph object with which the player interacts is displayed in the center of the screen. On the left hand side of this object you can see the connectivity of the given architecture. Displayed below are the original circuit before the users optimisation, and a circuit optimised by the user. Finally, scoring information and circuit statistics are shown at the top of the page.

4.4 How to Play

Upon accessing the game, the user is provided with a level selection choice. The circuits that are currently built into the game are; the Greenberger–Horne–Zeilinger (GHZ) state generator, an implementation of a Quantum Error

Correction code for the 3 and 5 qubit cases, the Quantum Fourier Transform (QFT) for a number of qubits, and the 2-qubit Grover Search algorithm (GSA). Such circuits are likely to be widely used in quantum computing, and as such finding an efficient form for each of them for a multitude of different architectures will likely be extremely useful.

Three of the ZX rewrite rules, discussed in Section 2.1 are available to the user as moves. These are:

- Colour Change

  - This is implemented by double clicking on a spider and has two effects. The selected node colour is swapped and all black wires connected to the spider are replaced with a blue wires and vice versa.
  - Figure 7a (a) shows a red spider with a blue wire connected to the right. By applying a Hadamard on both wires and changing the colour to a green spider, Figure 7a (b) is recovered as when a Hadamard is applied onto a blue wire, a standard black wire is recovered.

- Spider Fusion

  - When two spiders of the same colour are directly connected by a black wire they can be fused. The Spider Fusion rule is applied by dragging one of the spiders onto the other.
  - This is depicted in Figure 7b, where the fusion of two red spiders with 0 phase on each spider. If either spider has a phase on it, then the phase of the fused spider will be the sum of the two.
  - The inverse of the Spider Fusion is Slicing. This is applied by drawing a line across the spider, which produces two spiders of the same colour connected by a black wire. The sum of the phases of the two new spiders equal the phase of the initial spider.
  - Figure 7c depicts a slice to produce two spiders. Subfigure (a) shows the specific slice used to create the two spiders present in Subfigure (b). If the slice was at $90°$ to the depicted slice, the position of spiders 9 and 13 in Subfigure (b) would be flipped.

- Spider Identity

  - If a spider, with no phase, is connected in a diagram with only two wires connected, then it can be treated as an identity. This rule is implemented by a single click on the spider. This can also be used to place a spider with 0 phase onto any black wire within a diagram. This can be seen in Figure 7d

One further rule that has been added into the game which is not formally in the set of ZX rewrite rules, but is still compatible with the ZX graph formalism. This rule enforces a specific input state, resulting in graphs which are inequivalent under the ZX rewrite rules. However in certain circumstances where the input state is fixed, i.e those of resource state generator circuits, this rule can give more flexibility in performing certain circuit transformations. Potentially resulting in a more optimised circuit than can be accessed via only the ZX rewrite rules.

- Qubit to Spider

  - This rule is implemented by a single click on a starting node which produces a red zero phase spider, corresponding to an input of $|0\rangle$ for that qubit. This is displayed in Figure 7e. This allows circuits which depend on specific inputs, such as Grover Search and GHZ state generation, to be simplified further than for only arbitrary inputs which are not needed in these specific cases.

One example of how a level in our game can be completed is given in Figure 10 where the GHZ circuit is depicted. The level starts in Figure 8a with the basic GHZ state on a architecture that is a single line of qubits. This Figure shows that the initial ZX graph would require a large number of swap gates to be implemented in its current form. As this circuit takes in a set of qubits that are initialised to $|0\rangle^{\otimes 5}$, the qubit to spider rule can be applied to pull four red spiders out of four of the nodes, as shown in Figure 8b. From this the four red spiders can then by fused with the other red spiders that they connect to via the spider fusion rule, as shown in Figure 8c. Now that the red spiders only have two wires connected and a phase of $0\,\mathrm{Rad}$ they can be treated as identities and the spider identity rule can be applied. This gives a graph as shown in Figure 8d.

This state can then be moved to better fit the architecture. By applying the Qubit to Spider rule the node of qubit 0 can be changed to a spider and then be converted back to a node at a position that better suits the architecture, as shown in Figures 8e and 8f. Using the Spider Fusion rule the five green spiders can be fused to

(a) An application of the Colour Change rule on part of the 2-qubit Quantum Fourier Transform.

(b) An application of the Spider Fusion rule on part of the 2-qubit Quantum Fourier Transform.

(c) An application of the Spider Fusion rule on part of the 2-qubit Quantum Fourier Transform.

(d) An application of the Spider Identity rule on part of the 2-qubit Quantum Fourier Transform.

(e) An application of the Qubit to Spider rule on part of the 2-qubit Quantum Fourier Transform.
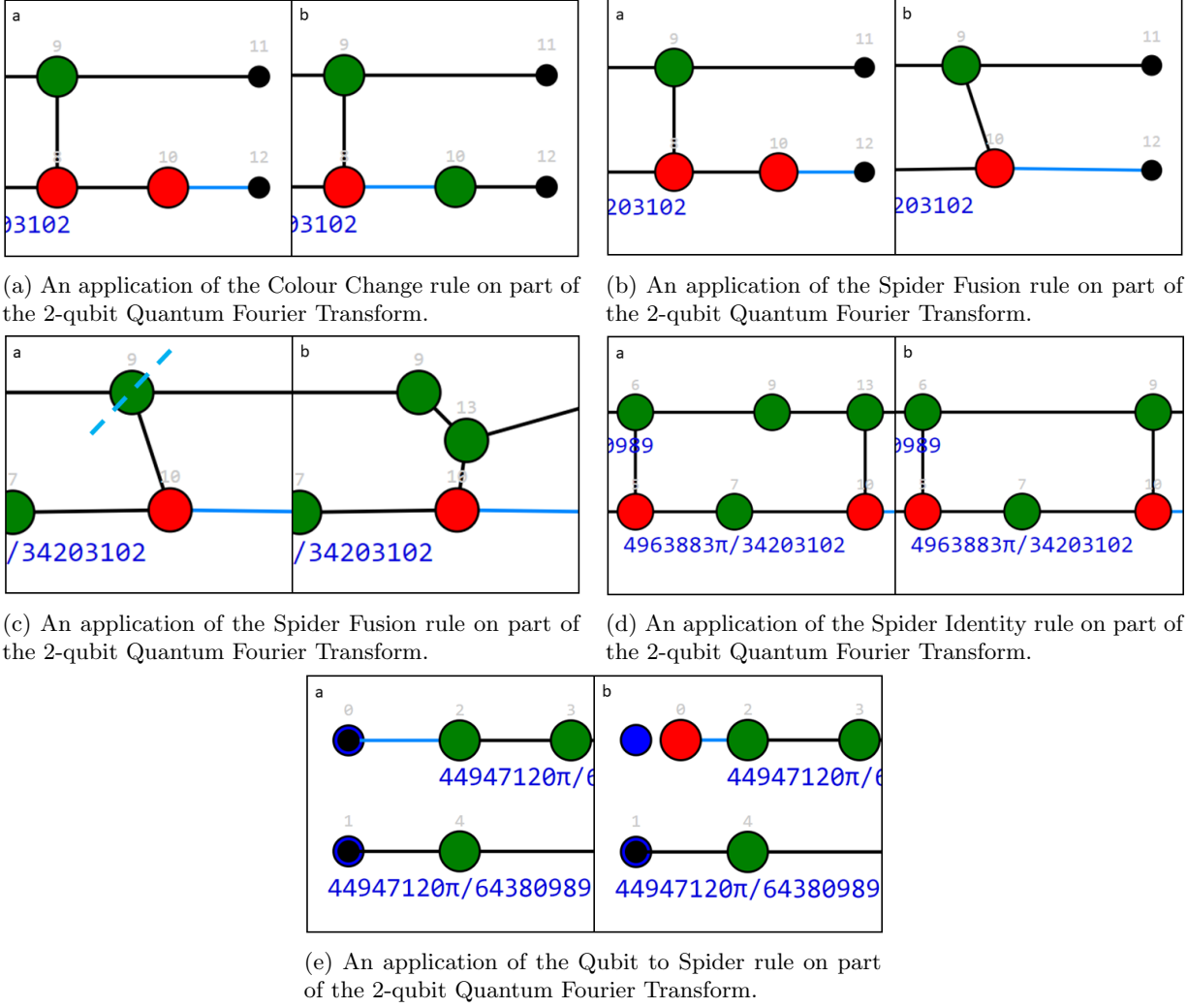
Figure 7: All four rules that are currently implemented within the game. These rules allow for simplification and optimisation of circuits, as the other rules have very small impact on the optimisation of circuits.

produce the diagram in Figure 8g. By applying two slices on the single green spider in Figure 8g two more green spiders can be made in the positions shown in Figure 8h. These slices were done such to slice along the path of qubit 2 and to produce green spiders that spread out in a column.

The central green spider can then be further sliced to separate this column of spiders into two columns of two spiders. Each of these two new spiders connect to two output nodes as shown in Figure 8i. The Spider Identity rule can then be used to place four red spiders into the diagram, in such a way that the new graph optimally fits the architecture. This can be seen in Figures 8j and 8k. The Spider Fusion rule can then be used to perform a slice on all of the red spiders in order reconnect the circuit back to the nodes, as shown in Figure 8l. This then produces the circuit that optimally fits the architecture, but only for an input state of the $|0\rangle^{\otimes 5}$. The output of this circuit and the initial circuit may differ if any other input state is used.

4.5 Architecture dependence in ZX-calculus

For any level, there are multiple ways to map the qubits to a physical architecture. This depends on the physical device used, and the subset of qubits chosen within the device. Examples of this can be seen when looking at the IBM Melbourne device and only needing to use 5-qubits, like for the GHZ level shown in Figure 6. The full device architecture for the 15 qubit Melbourne device is shown in Figure 9. On this architecture there are only 3 unique
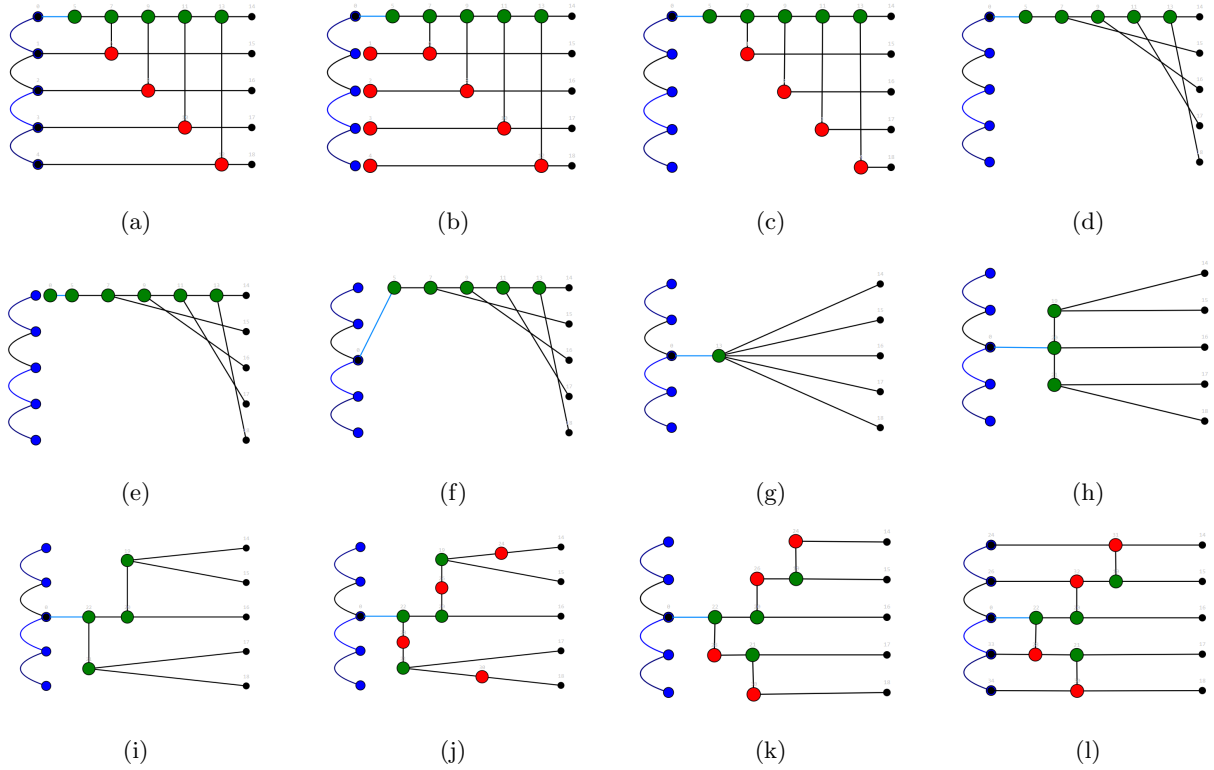
Figure 8: An example of the GHZ circuit played out within our game. This shows use of the rules depicted in Figure 5

combinations of qubits that can be combined to create the 5-qubit connectivity. These are; a line connected graph shown in Figure 10a, a T connected graph shown in Figure 10b, and a loop connected graph shown in Figure 10c. The T connected graph is defined as any structure that is not a line connected graph or a loop connected graph. Each of these will have a different solution to the game as the structure of the most efficient circuit will change depending on the architecture.
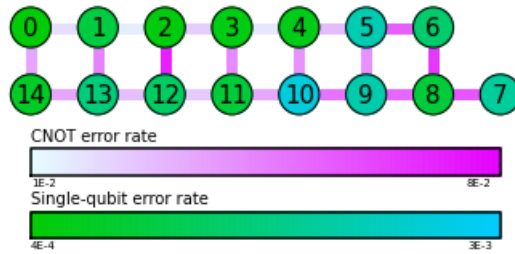
Figure 9: The full connectivity graph of the IBM Melbourne device. This shows the error rate of both single qubit gates and of the CNOT gates between qubits.

4.6 Scoring

Scoring the Zedex game proved to be one of the trickiest tasks faced. This is almost entirely due to the how hard it is in general to extract a circuit from a ZX graph object [?]. In the following we discuss both our methods of circuit extraction, and methods of scoring based on the extraction output.

12
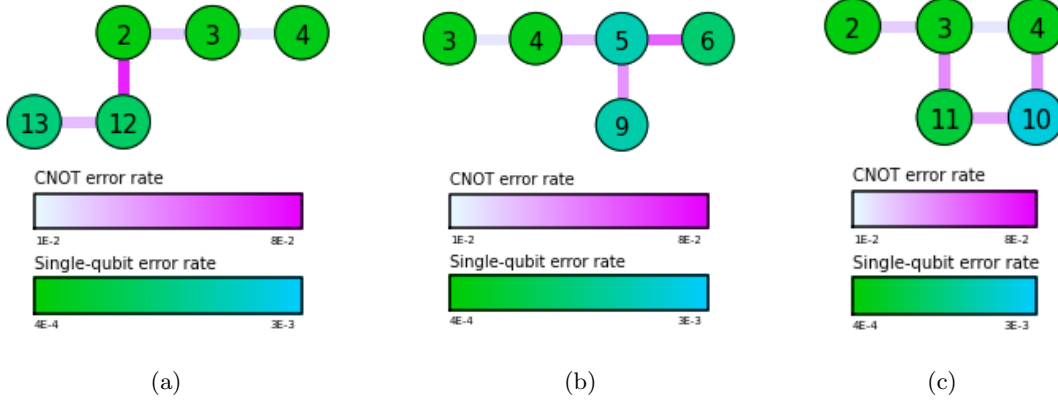
Figure 10: The connectivity graphs of 3 possible sub sets of the system shown in Figure 9. The first graph, shown in Figure 10a, is a line connected graph that is the same architecture that is used in the GHZ level example shown in Figure 10. The second graph, shown in Figure 10b, shows a T connected structure. This is a line connectivity with another single or line set of qubits connected to one qubit in the original line. The third graph, shown in Figure 10c, shows a loop connectivity. This is essentially a line connectivity that is folded back on itself to include more connections between qubits.

4.6.1   Extracting a Circuit to be Scored

The bulit-in extraction algorithm requires that the ZX graph be in semi-normal form before it can be efficiently extracted. The pyZX full_reduce (full optimisation) method returns a graph in semi-normal form which as a result can be extracted. In circuits composed soley of Clifford gates, the semi-normal form corresponds to having every internal edge as a hadamard edge, and every node a green node. In general, circuits contain non-Clifford gates which results in significant complications in circuit extraction. Converting non-Clifford circuits into semi-normal form involves the application of phase gadgets, which are non-intuitive, and discard the architecture dependence introduced for the game. As such they (currently) cannot be implemented as a rule resulting in graphs, which are not in appropriate form, and the circuit extraction fails. This presents a significant challenge for our game, as any any circuit which provides a quantum speedup must be non-Clifford [?]. In the Clifford case we have developed a post-processing algorithm, which takes the users graph to semi-normal form though appropriate addition of identities, described below.

- Take the zx graph and use to_gh() to turn all spiders green

- Insert combinations of spiders and Hadamard edges that correspond to identities, in order to turn all internal edges to Hadamard edges.

- Use extract_circuit() and turn to a qasm string

- Loop through the qasm string and delete any double hadamards

- Compare the tensors of the original pyzx graph and this final qasm string. If True, display the circuit.

In this way we can display an optimised output circuit for the Clifford case. To extend this to all circuits, we are communicating with the authors of pyZX to work out if it is feasible to convert a non-Clifford circuit to semi-normal form whilst retaining the architecture dependence. This would allow us to extract circuits successfully in all cases, whilst maintaining the users optimisations. Having a successfully extracted circuit is desirable, both in order to give the user instant feedback on their optimisation, and in order to 'score' their efforts.

4.6.2   Scoring Methodology

We desire that the best score in our game corresponds to a circuit that is maximally optimised for given architecture. The ultimate measure for this would be to run the circuit on a quantum computer and compare the actual fidelity to the ideal case. In practice measuring the fidelity is infeasible but one can sample from distributions in order to approximate the fidelity. In particular, if a quantum circuit is simulated N times and then the same

circuit is run N times on a real quantum machine, the distribution of the results in each case should be similar. As the impact of computational errors increases, the difference between these distributions should increase as well. Therefore, the divergence between these two distributions can be a practical estimation of the fidelity of the circuit. Such metrics are the Kullback-Leibler divergence (D_{KL}) [20] and the Jenses-Shannon divergence (D_{JS}) [21]. The use of such metrics, and especially of D_{JS}, is described and used in [22] as a method to benchmark their compiler. To use this as a metric would require in-browser integration with `Qiskit` which proved technically difficult.

In the absence of `Qiskit` integration to measure the fidelity, there are a number of factors that would contribute to a fully optimised circuit that we would like to consider:

- The routing problem: Do the control gates in the circuit align with the underlying connectivity of the architecture?

- Circuit depth: The number of gates on each qubit.

- Two qubit gate count

- The fidelity of the control operations between given qubits on the architecture

- T-gate count

If the two-qubit gates do not lie on the architecture then, one or more SWAP gates will be applied so that the gate between them can be executed. This depends on the way the compiler has allocated each qubit. In general, SWAP gates degrade the fidelity of the circuit, as for IBM's machines they correspond to three CNOTs on the physical qubits. So, counting the total number of SWAP gates that are needed could work as a metric for achieving equivalent circuits with higher fidelity.

However, since the complete routing of the qubits is not accessible, the scoring in this case will be an approximation of that number. To find this approximation, we convert our architecture to a weighted graph object []. Here the qubits are nodes, and the connections are represented by weighted edges, where the weights are the fidelities of the connections. Based on this, the game initially uses Dijkstra's algorithm [23] to find the lowest total weight path between two qubits. We wish to calculate the number of SWAPS to traverse this path, and then return to the initial position. While not efficient this ensures we always make valid SWAPS. Since the SWAP gates are applied twice, the game computes the double value of the error rate of all these SWAP gates. This error rate associated with the swapping is then added to the inherent error rate of the two qubit gate. This is repeated for every two qubit gate for which the input qubits are not connected in the architecture, and all the individual error rates are added to form a final "routing error rate". It should be noted that as a method of solving the routing problem to generate an optimised circuit, this would be sub-optimal. However all we are trying to obtain is a cumulative measure of how far the users two qubit gates are from matching the architecture. This means all we need is a consistent metric by which to assess the user.

When two-qubit gates are on the architecture, they are still subject to errors characterised by the gate fidelity. By multiplying the number of uses of each two qubit gate with its associated error rate, we get the final "two qubit error rate". In addition, each single qubit operation inserts a gate specific error rate to the computation. By multiplying this rate with the number of single qubit gates on each qubit, we get a final "single error rate". Finally, we measure the total error rate of the circuit by adding a weighted version of the routing, the two qubit and the single error rates. Ideally we could inform these weights by comparing to the circuit fidelities discussed above. The final score is then computed by subtracting the total error rate from 100%.

4.6.3 Scoring when circuit extraction fails

If the circuit extraction fails, finding a metric by which to score the users efforts becomes difficult. Fortunately even then, the 2 qubits gates do not get switched around, meaning they are on the qubits the user intended. This means we can still provide a shortened version of the score based on this type of gates. In particular we can calculate the "routing error rate" and the "two qubit error rate". Then we can follow the same procedure with rescaled weight values for this case and calculate the final score. We also display an error message to ensure the user is aware the optimised circuit is not valid.

4.7 Different types of level

During development it was realised that different circuits have either there own specific strategy or serve a specific purpose within the game. Here we present these classes of levels and what purpose they serve.

4.7.1 Randomly generated Clifford circuits

We have levels that are randomly generated and only comprise of Clifford gates. These circuits don't have an application in of themselves but we found that they can be good 'toy' example levels that can help the play get familiarity with the rules of the ZX calculus. This is due to the circuits being fairly simple with lots of opportunities for spider fusion and colour changes to reduce the number of nodes in the graph. These circuits are also easier to extract, and so in these toy levels it is possible to have a before and after circuit display to the player that will update with each move, giving the player real time feedback. This could help a player familiar with quantum circuits see the connection between the ZX rewrite rules and their corresponding transformation on the circuit. Architecture specific optimisations for these circuits are challenging due to simple structure of the circuit and reduced set of rewrite rules available to the player. For this reason scoring based off of architecture connectivity is not effective. The alternative scoring method using circuit depth an gate count is more applicable for these levels, and gives the player a concrete task to perform by reducing the overall depth and gate count as much as possible.

4.7.2 State generator circuits

Another class of level featured is the optimisation of state generator circuits, where our flagship example is the 5 qubit GHZ circuit. We define state generator circuits as those which take the all zeros state to some resource state. The key attribute for these circuits is that they always take in the all zeros state as input, and we can utilise this for a game mechanic. If the input to a circuit is the all zero state then this corresponds to to input qubits being red spiders with zero phase. These spiders can then interact with other spiders with the rewrite rules and so long as red spiders an be brought back to the inputs then the rewritten circuit will generate the same state as before. The overall unitary will not be the same in general however. We have found no indication in the literature of the ZX calculus being used in this way to optimise a state generator circuit. Future work would be to find other examples of useful state generator circuits, perhaps for the generation of cluster states.

4.7.3 General circuits for real world applications

This would be the theoretically most useful type of level. Examples we have implemented would be the quantum Fourier transform up to 15 qubits, and Grovers algorithm up to 4 qubits. The game effectively acts an an architecture aware high level circuit optimiser. However unlike in the state generator case we cannot use the mechanic of setting the inputs to red spiders. We believe that with the current subset of rules implemented that meaningful optimisation of these circuits is not possible. The other remaining rules would have to be implemented so that the full power of the ZX calculus is available to the player. Other compound rules such as local complementation and pivoting, both of which are available in pyzx, would most likely be required as well. The more complex nature of the remaining rules makes intuitive implementation more difficult, and is left for future work.

4.8 Outlook and Further Work

We hope that the game brings some intuition and visualisation to the many different factors that have to be considered in circuit optimisations. In this sense we hope that we have fulfilled our system requirement that the game be educational, although for a slightly more advanced audience. At the moment, the game certainly does not produce anything useful, and it is likely that it would not produce anything useful even if it had the full functionality we desire. However, we do think there is a small chance that giving the users the ability to manually control the two-qubit layouts simultaneously with gate operations, could yield optimisations that current compilers may miss. Until we can properly extract circuits this is not possible to test, and as such we have not fulfilled this system requirement. Whether the game is enjoyable or not is subjective, but we feel like the mechanics are intuitive and fun and so consider this requirement fulfilled. The game is available simply by visiting a webpage, making it accessible, and the user could take the qasm string for their optimised circuit and run it on real hardware.

There are a large number of future changes we plan to make to the game. First and foremost is coming up with a way to extract a circuit from the PyZX object that works for non-Clifford circuits. As discussed in Section 4.6.1, this has been the single biggest bottleneck to progress on the game so far. Once this is done, we would like to

optimise our scoring process, perhaps finding a way to integrate `Qiskit` so we can directly approximate the circuit fidelity. A fairly simple extension would be to allow the user to define their own circuit to work with, which would just require a qasm string of the circuit. Similarly user defined architectures would be a nice and reasonably simple extension to implement.

5 Squamble

The goal of Squamble was to take an interesting problem related to quantum theory and find a way to visualise it in an interactive and engaging way, with the main goal being to find a way to make quantum concepts more approachable and to provide insight into the underlying problem the game is based on. We chose a problem related to quantum walks as they can be easily represented on a graph making them ideal candidates for visualisation.

All code developed for the game can be accessed here: `https://github.com/naomisolomons/naomisolomons.github.io`, and can be played at: `https://naomisolomons.github.io/`.

5.1 Background theory

Consider a quantum circuit described by a unitary $\hat{U}$, with input $|\psi\rangle = \sum_x \alpha_x |x\rangle$. In this picture probability is thought of as static, existing in initial and final distributions. Instead of this it is possible to interpret this state evolution as probability flowing between computational basis states. If we define the flow of probability between the basis states n and m as f_{nm} then these flows must satisfy the following constraints:

$$f_{nm} \geq 0, \tag{1}$$

$$\sum_m f_{nm} = P_n, \tag{2}$$

$$\sum_n f_{nm} = P'_m, \tag{3}$$

where P_n is the probability of being at basis state n before the circuit and P'_m is the probability of being at basis state m after the circuit. These have been calculated from $\hat{U}$ and $|\psi\rangle$ via the standard quantum mechanical procedure. This corresponds to a matrix of positive values whose row and column sums correspond to the initial and final probability distributions respectively. These constraints mean the probability flows are not necessary unique, as any operation on the flows that leave the row and column sums invariant will give a valid solution. This is best illustrated with an example: consider a single qubit system. Then for any unitary and initial state, the flow matrix will take the form of a (2×2) matrix:

$$f = \begin{pmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{pmatrix}. \tag{4}$$

It is clear that transformations of the following form leave the row and column sum invariant

$$f = \begin{pmatrix} f_{00} + \alpha & f_{01} - \alpha \\ f_{10} - \alpha & f_{11} + \alpha \end{pmatrix}. \tag{5}$$

This shows it is possible to redistribute the probability flow around the computational basis states while representing the same circuit. This is just one of many possible ways of distributing probability a full description of the moves allowed in the current implementation of the game will be given later. It is clear that if the choice of α is unrestricted then this redistribution can lead to negative probability flow values which violate condition (1). Using the probability redistribution rules to remove negative values from the flow matrix forms the logical basis for Squamble as a puzzle game. In Squamble the user implements these rules by interacting with a graph where the nodes represent the computational basis states and the edges have weights corresponding to the amount of flow between those basis states.

The flow values presented in Squamble are calculated using the following formula

$$f_{nm} = Re(\langle\psi|n\rangle \langle n| \hat{U}^\dagger |m\rangle \langle m| \hat{U} |\psi\rangle) \tag{6}$$

developed in previous work by the author. This expression satisfies condition (2) and (3) but does not necessarily satisfy condition (1). This fact will aid in the development of random levels, discussed later. The aim of Squamble is two-fold: to illustrate these probability flows as an insight into the nature of quantum circuits, by representing there action in an engaging and visual way, and as a tool to investigate this formula further. We conjecture that any instance of negative elements in f_{nm}, as calculated by equation (6), can be removed via probability redistribution as described above. Therefore, a level that is not soluble with Squamble represents a counter example to this conjecture and thus disproves it.

5.2 How to play

The computational basis states are presented as nodes of a graph, in which directed edges between nodes show how the output is related to the input state: the labels of the edges show the 'flow' of probability between basis states, showing which output measurements become more likely, by 'receiving' probability from certain input states. For example, a Hadamard on one qubit (Tutorial Level 1) shows 0.5 flowing toward the basis state 1, and 0.5 flowing towards 0, from the input of 0 (as shown in Figure 11).
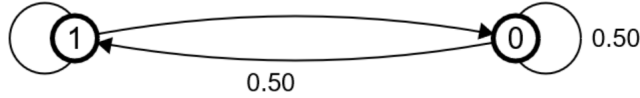

Figure 11: The representation of a Hadamard on one qubit in Tutorial Level 1.

The levels start with a particular quantum circuit (shown in Figure 12) illustrated with one of these diagrams. It is possible to choose any (appropriately connected) cycle of nodes in the graph, and rearrange the probability flow within this cycle, so that the edge values in one direction around the cycle increase, and decrease in the other direction. However, the result of the redistribution represented (the output of the circuit) does not change, as the total amount of probability going into or out of a node does not change.
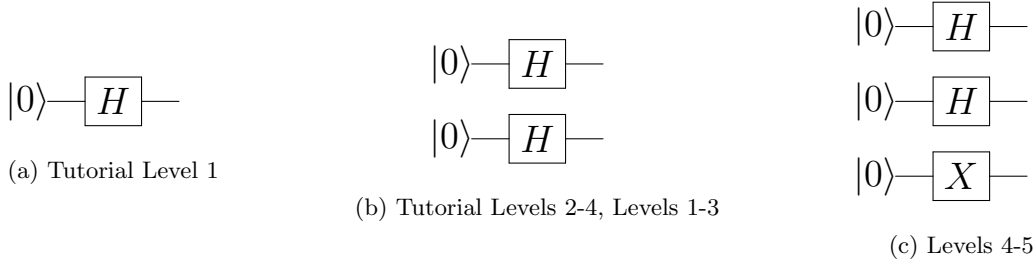

(a) Tutorial Level 1

(b) Tutorial Levels 2-4, Levels 1-3

(c) Levels 4-5

Figure 12: The underlying circuits for the current Squamble levels.

By pressing the 'Squamble' button, the values mix (to pre-decided values that are determined using this method of redistributing around cycles), so that some edge values are negative. The aim of the game is to return to the original distribution, or find another solution so that all edge values are non-negative. The theory discussed in Section 5.1 suggests that this is always possible, and finding a level that cannot be solved would disprove this theory.

The game can be played with the following steps:

1. Press 'Squamble' to change the edge values and set up the problem.

2. Choose the nodes of the cycle you want to rearrange by pressing shift and clicking them.

3. Press Check to confirm this is a valid cycle.

4. Scroll (using the mouse or trackpad, although there seem to be some issues using the trackpad when running the game using Microsoft Edge) to redistribute the probabilities around the cycle.

5. Press Submit.

6. Repeat steps 2-4 until all edge values are non-negative.

The game currently has a beginning animation when 'Squamble' is pressed, that shows how the probability is mixed at the beginning of the level (like showing a player how a Rubik's cube has been rearranged before solving it). This makes the puzzle considerably easier, and is a feature we could remove to increase the difficulty.

5.3 Puzzle Analysis

5.3.1 A Group Theory perspective

In this section we use group theory to formally define the mechanics and goal of Squamble. We defined a valid move in the game as an operation on the flow matrix which leaves its row and column sums invariant. It is clear that operations of the following form satisfy this

$$f + m \quad : \quad m \in M, \tag{7}$$

where M is the set of matrices that satisfy the following conditions

$$\sum_i m_{ij} = 0, \tag{8}$$

$$\sum_j m_{ij} = 0. \tag{9}$$

We will call this set M the move set. Conveniently this set forms an abelian group under the operation of matrix addition. We will prove that $(M, +)$ is a group by showing that it satisfies all the standard group axioms

1. Closure

$$\text{Let } m, m' \in M \text{ consider } m + m',$$
$$\sum_i (m + m')_{ij} = \sum_i m_{ij} + \sum_i m'_{ij} = 0 + 0 = 0,$$
$$\sum_j (m + m')_{ij} = \sum_j m_{ij} + \sum_j m'_{ij} = 0 + 0 = 0,$$
$$\implies m + m' \in M.$$

2. Associativity - Matrix addition is associative.

3. Identity element - The additive identity element is the all zeros matrix, $\mathbf{0}$, which clearly satisfies (8) and (9). Hence, $\mathbf{0} \in M$

4. Inverse element - For each element $m \in M$ the inverse can be constructed by element wise negation of m.

$$(m + (-m))_{ij} = m_{ij} + (-m)_{ij} = 0, \ \forall \ i, j,$$
$$\sum_i (-m)_{ij} = -\sum_i m_{ij} = 0,$$
$$\sum_j (-m)_{ij} = -\sum_j m_{ij} = 0,$$
$$\implies -m \in M.$$

5. Commutativity - Matrix addition commutes.

This allows us to think of applying a move to the flow matrix as the group M acting on the set F, where F is the set of all matrices satisfying condition (2) and (3). Formally we can write this action as a homomorphism, φ.

$$\varphi : M \times F \to F \ , \ \varphi(m, f) = f + m = f' \in F. \tag{10}$$

Therefore the goal of the game is to find a $m \in M$ such that $\varphi(m, f) = f_s, \ f_s \in F_s$. Where F_s is the set of matrices satisfying conditions (1), (2) and (3), this is known as the solutions set.

Knowing that this underlying structure exists is important for showing that Squamble works as a puzzle game. From this group structure we can see that given $f_s \in F_s$ that gets mapped to some $f \in F$ (Squambled) we will always be able to find $m \in M$ that takes us back to the solution space. The property of closure is also useful as it means that simpler moves can be combined to make more complex moves, meaning that the player can build up algorithmic approaches to solving a level. The addition of commutativity is a bonus as this means the player is not restricted to performing moves in any particular order to reach a desired outcome.
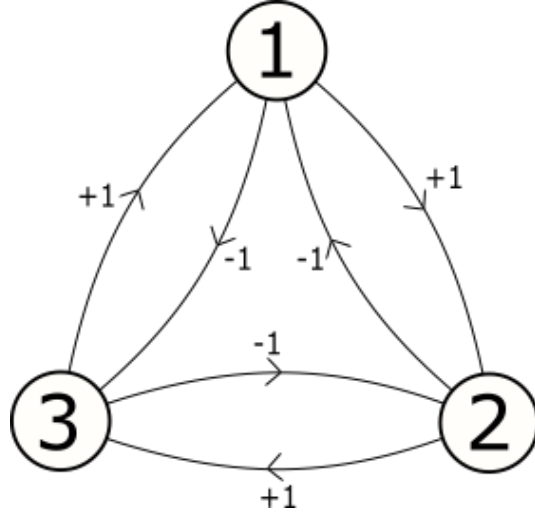
Figure 13: Showing the graphical representation of a basic move.

5.3.2 Currently Implemented Moves

The moves from M that have been implemented into the game in its current form fall into two categories, basic moves and advanced moves. Basis moves are easy to visualise and can be selected intuitively via the games interface, while advanced moves require a deeper understanding of the move set and have to be selected via a less intuitive user interface. The special structure of these moves will be discussed below.

Basic Moves - These moves can be easily visualised as redistribution of probability around cycles. For example consider

$$m = \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix}. \tag{11}$$

This move can be represented on this game graph and is shown in Figure 13. The representation of adding flow around one loop and removing from the other makes this move easy to understand. It is easy to see that this move can be expressed in the following way

$$m = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}^t. \tag{12}$$

This representation motivates the following definition for a basis move. A move, $m \in M$ is called basic if it can be represented as $m = \alpha(P - P^t)$, where α is some real number and P is a non-symmetric permutation matrix. Due to the simple structure of these move they can be implemented simply in the game. For example to perform a basic move between nodes (1,2,3), the user selects those nodes and clicks the check button. This then allows this move to be performed via the scroll wheel on the mouse, or with the arrow keys.

Advanced Moves - These moves are less easy for visualise. For example consider the following move

$$m = \begin{pmatrix} 0 & 1 & -1 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix}. \tag{13}$$

Which is shown represented on the game graph in Figure 14. This shows how these advanced moves are less intuitive as it is not immediately clear from the graph that they leave the underlying flow matrix invariant. The game has been designed to allow the user to implement any move where the only non zero elements form a rectangle. More formally, we say a move $m \in M$ is called advanced if the only non-zero elements of the move matrix are of the form $m_{ij}, m_{(i+n)j}, m_{i(j+m)}, m_{(i+n)(j+m)}$ for some $n, m \in \mathbb{N}$. To execute an advanced move in the game the user selects the nodes that wish to represent i and j, then they click the advanced button (This can be accessed by pressing shift + A). The user will then by prompted to enter a value for n and m. If the input is valid then the move will be accepted.
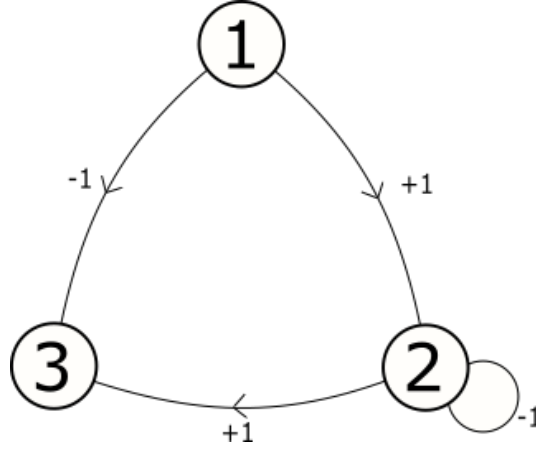
Figure 14: Showing the graphical representation of an advanced move.

5.4 Level Design

Tutorial Levels: The beginning levels of Squamble are designed to introduce the mechanics of the game to the user. Hence, the mixing of each of these levels is very simple with only one or two mixing moves per level. Each of these levels also have a tutorial displayed in the top third of the screen.

Main Levels: The bulk of the levels currently designed for Squamble build of the moves explored in the tutorial section. Now each level has a more complex Squambling process. Meaning more care needs to be taken when solving the puzzle. The final two levels have been implemented on a 3 qubit graph. This graph consists of 8 nodes showing how adding a single qubit doubles the amount of available basis states. The flows for all these levels are calculated using equation (6). Interestingly these calculated flows are positive showing that this formula for the flows does generate valid solutions for simple unitaries and initial states.

Randomly Generated Levels: The final level is randomly generated. This is done using `Qiskit`'s random circuit generator. Due to this dependence on `Qiskit` the level has to be updated manually as `Qiskit` can not be run in the browser. This level is probably the most interesting as it tests the users understanding of the games mechanics and potentially requires the advanced move set to be solved.

5.5 Analysis/Systems requirements

We consider the game with regards to the systems requirements, as discussed in Section 3.

Educational: the way the levels are displayed gives a visualisation of the way that probability 'flows' in a quantum circuit; that is, the probability of a certain outcome when measuring in the computational basis at the output of a circuit, compared to the input state. The prepared levels are shown in Figure 12 and are designed to illustrate simple quantum gates.

More complicated circuits were considered, but implementing these proved difficult. Firstly, the visualisation does not show phase, and therefore representing, for example, a CZ gate or quantum Fourier transform, is not possible (or distinguishable from multiple Hadamards). Secondly, the size and complexity of the graph means that it is not practical to represent large circuits, and we avoided any circuit with more than 3 input qubits. Finally, representing some circuits (for example, generating a GHZ state) was possible, but proved to give a simple or disconnected underlying graph structure that wasn't suitable for level building. Therefore, the educational aspect is limited by the number of circuits that can be appropriately displayed - however, the user should gain a more intuitive understanding of the result of certain gates.

Useful: the game was originally developed with the goal of collecting examples to investigate the ideas discussed in Section 5.1. The most useful outcome in this case would be a level which is insoluble, and therefore demonstrates that our earlier conjecture is false.

However, the game is currently set up with the majority of the levels associated to previously generated puzzles. Useful data comes from players generating and playing random levels. It is also unknown whether they would come across an insoluble level at any point, and hence generate a useful output to the game. However, playing these levels my lead to the development of puzzle solving strategies, which in turn could give insight to the underlying problem.

Enjoyable: the tutorial levels, colour coding system, and beginning animations were all added to make the game clear and easy to understand. The user receives immediate feedback from the game and the scoring system keeps track of the users progress by counting the number of moves used to solve the puzzle. The randomly generated level adds more variety to the difficulty.

Feedback from the cohort?

Accessible: the game is available online, hosted by GitHub pages. It doesn't require any prior knowledge of coding or quantum mechanics to play, and starts with easier, tutorial levels. However, some further reading or background knowledge from the user may be necessary to make the connection to quantum circuits clear. The full code is also available to download for those wishing to get into the inner workings of the game.

Hardware: the game uses data generated by `Qiskit` to model the probability flows in circuits, however it is not specific to any particular hardware. It does not run using data generated by a quantum computer (it only models a theoretical circuit).

Therefore, the main objective in which Squamble succeeds, is to present a challenging and intuitive puzzle game that illustrates concepts of quantum mechanics, giving the user familiarity with a few particularly important gates. It may be of interest to more advanced users how the internal mechanics of the game are linked to more in depth theoretical work.

5.6   Outlook and further work

We could improve the interface of the game by looking at single qudit systems. This would allow us to have graphs with nodes of all integer values giving us polynomial scaling in the graphics rather than exponential scaling. However this would lead to other difficulties such as less educational relevance as qudits are not as prevalent in quantum information research. This would also require the back end to be completely redesigned as `Qiskit` is designed to work with qubits, though this could be beneficial as it may mean that the dependence on Python could be removed making integration into a fully web based platform much more straight forward.

Currently, there is the functionality for a random level to be easily generated and uploaded. This could be further improved by allowing users to generate their own levels. This would greatly increase the amount of data that can be generated. However, to achieve this we would need to host our game on a server so we could deal with data transfer requests. A further development goal would be to integrate the JavaScript library Q.js to give the user the tools to build there own quantum circuits in the browser.

The graph visualisation that Squamble uses could also be of interest as more a pedagogical tool to describe and understand certain quantum circuits. Further work could develop this in a way that could be used in other contexts. Additionally the representation of classical operations could be included to show contrast with the quantum operations.

6 QGame

(name to be changed)

QGame is designed around the qubit routing problem, with the dual goals of explaining the problem and providing inspiration and insight for future approaches. In short, the game is visually displayed as two overlapping graphs, one corresponding to the two-qubit gates in the circuit, and other to the architecture. The only move available to players is to swap nodes by clicking on them.

There are two main stages to the game:

- **Stage 1:** Swapping nodes here corresponds to relabelling the qubits of the initial circuit. Therefore these operations incur no penalty, and the player may do this as many times as they desire. Also, there are no restrictions on which nodes can be swapped, as all permutation relabels are possible.

- **Stage 2:** The two-qubit gates in the circuit are looped over. If a gate is currently lying on a native hardware connection, then the player can accept this and move onto the next gate. If a gate does not currently lie on the architecture, the player must swap nodes in order to be able to implement the gate and progress onto the next gate. Each swap corresponds directly to a SWAP gate (= 3 CNOT gates) between the relevant qubits, and as such incurs a penalty of 1. When all gates have been looped over, the game is finished.

The outputs of the game include the initial qubit placement, the final circuit, and the total number of swaps used, which is used as a scoring metric.

6.1 Under the Hood

One way of understanding how the game works is to view it as maintaining and updating a mapping between physical and logical qubits. The first stage is relatively intuitive to grasp: simply relabelling the qubits in the circuit will result in the same overall unitary being applied.

In Stage 2, however, things become less intuitive. Suppose we progress through the circuit and then find a gate that is not currently lying on the architecture. We apply swap gates until the gate now corresponds to a native gate. But now from this point on, all future gates in this circuit must be applied in light of this swap, otherwise the final unitary will be not be equivalent to the initial circuit.

6.2 Checking the Output

There are two main requirements of a candidate output circuit from the game.

1. The output circuit must only contain two-qubit gates that are native to the given architecture.

2. The output circuit must be equivalent to the input circuit.

The first requirement is clear: this is the whole purpose of solving the qubit routing problem. The second requirement is much more subtle, however. Equivalence between the input and output circuit is not immediately obvious.

Under stage 1 operations only, equivalence of output and input circuits amounts to a relabelling of the qubits, corresponding to a permutation basis change of the circuit unitaries. Hence the unitaries are related by $U_{out} = P_1 U_{in} P_1^{-1}$, where P_1 is the permutation matrix representing the initial logical to physical qubit mapping. In the second stage of the game, interstitial SWAP gates are inserted in to the circuit, such that the final and initial qubit mappings may no longer be equal. In this case, the output circuit corresponds to the input circuit, followed by a series of SWAPs, thanks to appropriate updates of the circuits, i.e. $U_{out} = P_2 U_{in}$. The game combines these two operations sequentially, so the general 'equivalence' of input and output circuit is given by the relation

$$U_{out} = P_2 P_1 U_{in} P_1^{-1}, \tag{14}$$

where the permutation matrices P_1 and P_2 are constructed from inital and final qubit placements.

6.3 Benchmarking the game output

In order to judge how well gameplay solves the qubit allocation problem, two benchmarks were identified. The simplest is the two-qubit gate count. For NISQ machines, two-qubit gates have relatively lower fidelity, so reduction of this metric is a useful way to mitigate computational errors. Shown in Figure 16 is the excess CX count.

23

The overarching goal of qubit allocation is to maximise the fidelity of the final circuit when run on a real (noisy) machine, so fidelity would be the ideal metric to evaluate performance. However, determining fidelity requires an exponentially large number of circuit measurements, making it unfeasible, so the Kullback-Leibler (KL) divergence was chosen to compare simulated and measured distributions.

6.4   Scaling of the game

The game has currently only be played up to 15 qubits. One advantage of the game design is that no simulation of the circuit is necessary, so the game could in theory be played at any number of qubits. The main memory requirements of the game lie with the circuit. Specifically, the game must keep track of the mapping from qubit labels in the original circuit to qubit labels in the current, modified circuit, and append new gates to the current circuit based upon this. Note that any quantum circuit must have a reasonable classical description, and so increasing the number of qubits would not pose a bottleneck in this regard.

However, attempts at benchmarking would fail as the qubit number increases beyond the limit of what is simulable. We would no longer have access to the perfect output statistics to compare the game output statistics with. The game could still be played but it would be unclear as to its effectiveness over alternative methods, such as the `Qiskit` transpiler. We believe that alternative methods would be preferable for a larger number of qubits, as for one playing the game would take much longer than heuristic approaches.

We also note that the playability of the game is expected to drastically decrease with both the qubit number and circuit depth. For large numbers, the gameplay will look like two overlapping highly complex graphs with many nodes, with the player only capable of swapping any two at a given moment.

6.5   Practicalities

The game is currently written in Python, mainly for ease of integration with `Qiskit`. The game displays as an interactive plot using the `matplotlib` visualisation library, from which the game registers clicking events and updates the plot accordingly.

There are two current options of playing the game. The first is to download or clone the repository, check that the necessary Python packages are installed, and run the levels directly via e.g.

```
python level_3.py.
```

The second option allows the player to play the game in their browser, without having to download anything. This involves the use of Binder [], which is a tool to host jupyter notebooks online from a github repository. Players can follow the link [] after some loading time, explore and play the game via an online jupyter notebook.

6.6   Analysis of System Requirements

We consider the game with regards to the systems requirements, as discussed in Section 3.

Educational: The game certainly serves as a possible tool of explaining the qubit routing problem. We believe that the simple visualisation of two overlapping graphs may contribute towards a conceptual understanding of how swaps may be necessary in order to run a circuit on an architecture. However, more testing of the game is needed, particularly by those with little background in the subject, in order to effectively determine this.

Enjoyable: This is a highly subjective criteria, and again could only be accurately assessed upon receiving feedback from a large number of players. We believe that the puzzle aspect to the game can be intellectually stimulating – we find that after a few attempts at playing one can get an intuition for which swaps may be the most beneficial. As the skill of the player increases, we anticipate that they will start to look ahead more to future circuit gates and try to tackle the problem as a whole, rather than by individual gates. It should also be said that we believe that as the number of qubits and circuit depth increases, the game will quickly become less fun. As the only move offered to the player is of single swaps, the number of swaps needed to achieve a reasonable result will likely increase exponentially, which could become monotonous. Perhaps more complicated moves could be added to combat this.

Accessible: The game could certainly be improved in this regard. Whilst we think the game is reasonably accessible to those in the quantum information community, or to those with programming experience, it is possible that the game may prove too confusing, boring, or pointless for a wider audience. The ideal situation would have involved hosting the game somewhere online, with limited loading time, improved visualisation and graphics, and a clearer explanation of the rules (perhaps with an interactive tutorial).

Hardware: The output of the game may be run on the `Qiskit` machines, and can be directly compared with alternative approaches. However, the game itself does not run on a quantum computer, and running the output on a real machine is considered an optional and bonus feature of the game. We felt that focusing the game too heavily around the real machines (for example, requiring the player to have an IBMQ account and to run something each time) would drastically reduce the accessibility.

Useful: It is still relatively unclear how useful this game may be. On the one hand, the ability the run the output of the game on real machines is certainly beneficial. Running any kind of job on real machines enables the owner to gather further information and diagnostics on the quality of their devices. This game may also serve to provide interesting insights into approaches for solving the qubit routing problem. If the game had many users, then solutions to the game could be directly compared against the state-of-the-art compilation approaches. If current methods are significantly outperforming the game (which perhaps is to be expected), then the usefulness of the game is limited to educational purposes and generating intuition for the problem However if the game can compete with or exceed current methods, then this could pose an exciting new way to improve current compilation procedures.

6.7 Further improvements

There several possible improvements that would substantially improve the game.

One is to improve the visualisation and hosting capability. The game may not be as easily accessible for those without experience of programming in python, and the online notebook version of the game can require substantial loading times. A preferred option would be to find a more natural way of hosting the game online, perhaps with the use of Python-friendly web frameworks.

A second improvement would be to explore further the link with `Qiskit` and to understand more precisely the role that the game could serve. At best, the game may provide a user-friendly way of directly engaging with and exploring the compilation process, and could lead to new and more intuitive qubit routing procedures. At worst, the game may be a useful tool in explaining the qubit routing problem and perhaps as a resource to teach students about quantum circuits and compilation.

There are also several smaller changes that may enhance the users gameplay experience. Not all of the available information is currently encoded into the game. for example, players do not know the order that the gates will arise. Including this information might enable the player to make more informed decisions if they knew which gates were coming up.

The game also is built around the univeral gate set of single qubit gates and CNOT gates – the game could be expande to include alternative gate sets. It also assumes that if a CNOT can be applied between two qubits, then the reverse gate (switching the target and control qubits) can also be applied – something that may not always be true in practice. Another small adjustment would be to include the fidelity of the connections, which could lead to more complicated scoring metrics.

7 Comparison

- Link all the games together

- Were we successful wrt the system requirements?

- If not, did some games fulfill certain aspects of the system requirements that other games did not?

8 Conclusion

The problems discussed in Section 2 proved to be appropriately challenging to use as a basis for puzzle based games that are specific to the hardware requirements of individual systems. However, it was difficult to present these problems in a way that would produce any useful output in the attempt to illustrate concepts that extend beyond the foundational ideas of entanglement and superposition may

In order to make the games widely accessible and reduce the time between moves, the goal of playing the game directly on a quantum computer was seen to be unrealistic, and was instead replaced by incorporating data from real devices into the gameplay. Future improvements with both the hardware and available software for quantum computers may change this, and so future quantum games may look very different. However, the games developed within the existing framework are fundamentally limited, and new game mechanics outside of what we have considered would be necessary to make full use of a large-scale, fault-tolerant quantum computer. The excitement of contemporary quantum games therefore arises simply from the interesting challenges and properties that are inherent to quantum computers themselves.

9 Acknowledgements

References

[1] J. Wootton, "Making games with quantum computers," https://medium.com/@decodoku/games-computers-and-quantum-84bfdd2c0fe0, 2020, accessed: 28-08-2020.

[2] M. Simmons, "Bertie the brain programmer heads science council," Ottawa Citizen, p. 137, 1975.

[3] K. Webb, "The \$120 billion gaming industry is going through more change than it ever has before, and everyone is trying to cash in," Business Insider, 2019. [Online]. Available: https://www.businessinsider.com/video-game-industry-120-billion-future-innovation-2019-9?r=US&IR=T

[4] J. Wootton, "The history of games for quantum computers," https://medium.com/@decodoku/the-history-of-games-for-quantum-computers-a1de98859b5a, 2018, accessed: 28-08-2020.

[5] ——, "Getting to know your quantum processor," https://medium.com/qiskit/getting-to-know-your-quantum-processor-ea418867615f, 2020, accessed: 28-08-2020.

[6] J. R. Wootton, "Benchmarking of quantum processors with random circuits," arXiv preprint arXiv:1806.02736, 2018.

[7] E. van Nieuwenburg, "Quantum tic-tac-toe," accessed: 28-08-2020. [Online]. Available: https://quantumtictactoe.com/

[8] J. Wootton, "Riding the quantum Ferris wheel," https://www.ibm.com/blogs/research/2019/03/quantum-ferris-wheel/, 2019, accessed: 28-08-2020.

[9] C. Cantwell, "Quantum chess: Developing a mathematical framework and design methodology for creating quantum games," arXiv preprint arXiv:1906.05836, 2019.

[10] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[11] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, "On the qubit routing problem," arXiv preprint arXiv:1902.08091, 2019.

[12] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in Proceedings of the 2018 International Symposium on Code Generation and Optimization, 2018, pp. 113–125.

[13] E. Jeandel, S. Perdrix, and R. Vilmart, "Completeness of the zx-calculus," 2019.

[14] B. Coecke and A. Kissinger, Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoni Cambridge University Press, 2017.

[15] R. Vilmart, "A near-optimal axiomatisation of zx-calculus for pure qubit quantum mechanics," 2018.

[16] B. Eastin and E. Knill, "Restrictions on transversal encoded quantum gate sets," Physical Review Letters, vol. 102, no. 11, Mar 2009. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.102.110502

[17] R. Duncan, A. Kissinger, S. Perdrix, and J. van de Wetering, "Graph-theoretic simplification of quantum circuits with the zx-calculus," Quantum, vol. 4, p. 279, Jun 2020. [Online]. Available: http://dx.doi.org/10.22331/q-2020-06-04-279

[18] A. Kissinger and J. van de Wetering, "Reducing the number of non-clifford gates in quantum circuits," Phys. Rev. A, vol. 102, p. 022406, Aug 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.102.022406

[19] M. Backens, H. Miller-Bakewell, G. de Felice, L. Lobski, and J. van de Wetering, "There and back again: A circuit extraction tale," 2020.

[20] S. Nishio, Y. Pan, T. Satoh, H. Amano, and R. V. Meter, "Extracting success from ibm's 20-qubit machines using error-aware compilation," J. Emerg. Technol. Comput. Syst., vol. 16, no. 3, May 2020. [Online]. Available: https://doi.org/10.1145/3386162

[21] A. P. Majtey, P. W. Lamberti, and D. P. Prato, "Jensen-shannon divergence as a measure of distinguishability between mixed quantum states," Phys. Rev. A, vol. 72, p. 052310, Nov 2005. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.72.052310

[22] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t|ket⟩: A retargetable compiler for nisq devices," Quantum Science and Technology, 2020. [Online]. Available: http://iopscience.iop.org/10.1088/2058-9565/ab8e92

[23] E. W. Dijkstra et al., "A note on two problems in connexion with graphs," Numerische mathematik, vol. 1, no. 1, pp. 269–271, 1959.

[24] "IBM quantum computing." [Online]. Available: https://www.ibm.com/quantum-computing/

[25] L. Sun, L. DiCarlo, M. Reed, G. Catelani, L. S. Bishop, D. Schuster, B. Johnson, G. A. Yang, L. Frunzio, L. Glazman et al., "Measurements of quasiparticle tunneling dynamics in a band-gap-engineered transmon qubit," Physical review letters, vol. 108, no. 23, p. 230509, 2012.

[26] "IBM Quantum Experience." [Online]. Available: https://quantum-computing.ibm.com/

[27] "Amazon Web Services." [Online]. Available: https://aws.amazon.com/braket/hardware-providers

[28] "Rigetti Quantum Cloud Services." [Online]. Available: https://www.rigetti.com/qcs/docs/intro-to-qcs

[29] "pyquil." [Online]. Available: https://pyquil.readthedocs.io/en/latest/start.html

[30] "IonQ." [Online]. Available: https://ionq.com/company

[31] "D-Wave." [Online]. Available: https://www.dwavesys.com/

A   Quantum Hardware and Software

Within the world of Quantum Computing there are currently four main providers. These are IBM, Rigetti, IonQ, and D-Wave. Each of these have different architectures, access requirements, and languages. The different providers of each are listed in Table 1

Provider	Type of Qubit	Access	Code	Assembly Language	Open Access
IBM	Superconductive	IBM QE	qskit	QASM	Yes
Rigetti	Superconductive	AWS	pyquil	Quil	Unknown
IonQ	Trapped Ions	AWS	Unknown	Unknown	Unknown
D-Wave	Quantum Annealing	AWS	Ocean SDK	Unknown	No

Table 1: The Quantum Computing providers, along with the information on accessibility and usability.

A.1 IBM

In recent years IBM has been one of the leaders of open-access quantum computing [24]. IBM's quantum processors are made up of superconducting transmon qubits [25]. The IBM system is accessed through IBM Quantum Experience (IBMQ) that takes in `Qiskit` code [26]. There are currently 9 backends that are available through IBMQ, as shown in Table 2 .

Computer	Qubits
Ibmq_16_melbourne	15
Ibmq_london	5
Ibmq_burlington	5
Ibmq_essex	5
Ibmq_ourense	5
Ibmq_vigo	5
Ibmq_5_yorktown	5
Ibmq_armonk	1

Table 2: The Quantum Computers that are available through the IBMQ user portal.

A.2 Rigetti

Althoug Rigetti is a leader in the development of functioning quantum computations, there is currently a lack of information regarding access. The computers may soon be accessible through Amazon Web Services (AWS) Braket [27], but there is still little information available without a paid for access. Rigetti computers are built using superconductive qubits. Their Aspen chips have connectivity graphs that are octagonal with mostly 3-fold connectivity.

Rigetti has a system called Quantum Cloud Services (QCS) [28]. QCS runs of Rigetti Forest SDK, which uses pyQuil, QVM and the Quil Compiler [29].

A.3 IonQ

IonQ is in a similar position to Rigetti, they are a leader in developing the technology but have little public information on accessing the computer. Unlike Rigetti there is no software available for designing circuits for this system. IonQ is also listed as part of AWS Barket but there is no information listed on IonQ about future access.

IonQ utilises trapped ionised ytterbium atoms with semiconductor-defined electrodes on a chip. Laser pulses are then used to prepare the the initial state and to apply the gate on the qubits [30].

A.4 D-Wave

D-Wave uses Quantum Annealing to solve problems represented as mathematical functions [27, 31]. This system uses cryogenic refridgeeration and sheilding to create an internal high-vacuum environment, with temeratures of mK, that is isolated from external stimuli. These systems will only accept problems that map onto the Quantum Ising Model, and are accessed through AWS by use of D-Wave Ocean SDK [31].
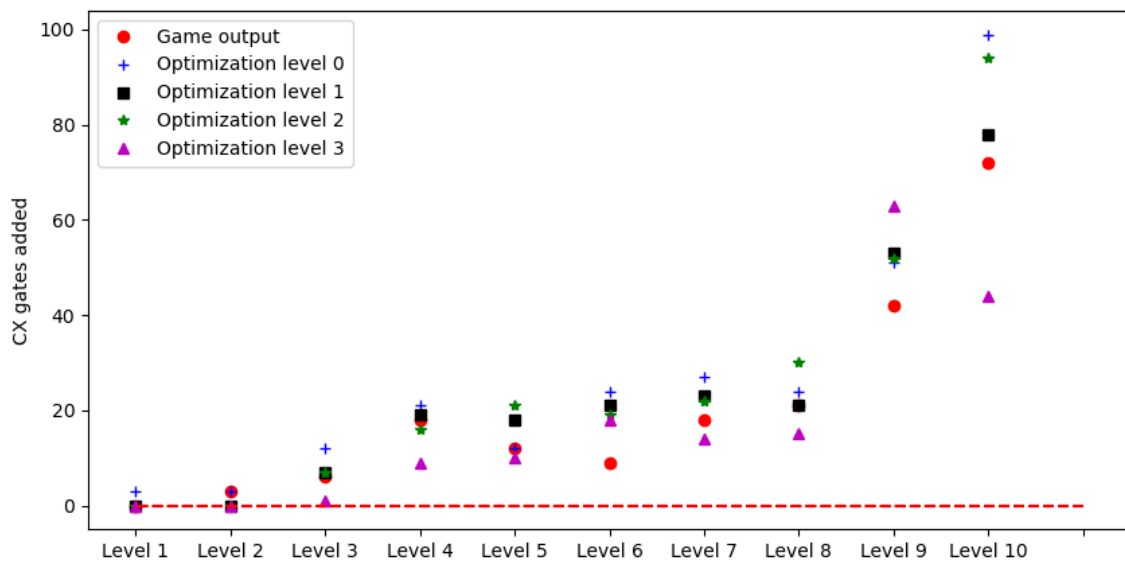
4

3

Architecture connection
Circuit gate
Next gate

29

Figure 16: Caption