physicsworld

QUANTUM COMPUTING FEATURE

Playing games with quantum computers

03 Dec 2018

Taken from the December 2018 issue of *Physics World*, where it appeared under the headline "Game on". Members of the Institute of Physics can enjoy the full issue via the *Physics World* app

Creating games for quantum computers offers an engaging way of exploring and testing their capabilities, writes James Wootton

Buying a computer typically involves staring long and hard at lists of specifications. As you try to weigh up processor speed against RAM, or hard-drive capacity against screen size, you may find yourself imagining what you might use the machine for, and how well each device might serve those needs.

This is a problem we are currently facing with quantum computers. New prototype devices are being announced every few months. Each time, it is the number of quantum bits, or qubits, that grabs the headlines, but figuring out how well the qubits work, and what they might be useful for, is not as easy to quantify. Often, it means going back and staring long and hard at lists of specifications.

But this isn't the only way to get to know a quantum device – we can also try them out. We can run programs that push their capabilities to the limit and give us relatable ways to understand their performance. Then we won't just have a list of numbers; we'll have experience.

To decide what kind of program to run, let's look back to the early days of digital computing. In 1961 scientists at the Massachusetts Institute of Technology (MIT) received a new model of computer: the PDP-1. Before it was even installed, people were already trying to figure out how to use it, and what they were going to do with it. Three researchers in particular – Steve Russell, Martin Graetz and Wayne Wiitanen – decided that they wanted to create a program that could do three things: push the device to its limits, behave differently each time it ran, and operate in the form of a game.

The game they made was called *Spacewar!*, and it was the first computer game to be more than just an expensive version of an ordinary board game. Players began with one spaceship each, both of them perilously close to their local star. Their first challenge was to fight against the star's gravity well. Then, once they had achieved something close to a stable orbit, their job was to hunt down and destroy their opponent.

The game did more than just give players experience with the PDP-1 – it also gave them an insight into orbital mechanics. They soon learned that gravity is not a force you can easily run away from, but one you have to work with. Developing a winning strategy meant working out what kind of orbit you wanted and how to achieve it. This was the first example of a concept we've seen many times since: games that offer people the chance to play with and learn about physics that is outside their daily experience.

Gaming the system: (above) *Spacewar!* running on a PDP-1 at the Computer History Museum in California. (below) Steve Russell, one of the game's co-creators, demonstrates the original *Spacewar!* controller. (Courtesy: CC BY Joi

Ito)

(Courtesy: CC BY Joi Ito)

These aspects of *Spacewar!* are exactly what we need now for quantum computers. We need programs that serve as examples of what a program can be, and that allow new users to learn by experimenting with the code. We need programs that will take full advantage of a device's capabilities and demonstrate its strengths and weaknesses. And we need programs that enable users to experience an otherwise inscrutable area of physics directly; to learn how it works; and to figure out how it can be harnessed.

> We need programs that enable users to experience an otherwise inscrutable area of physics directly; to learn how it works; and to figure out how it can be harnessed

Quantum battleships

This philosophy is part of what motivated me to start making games that run on quantum computers. After a few initial experiments, my first proper game was called *Battleships with partial NOT gates*. Like more traditional versions of Battleships, mine is played on a grid where each point represents a place where a ship might be hiding (figure 1). The grid is tailored to suit the device used to play the game. At the time I created it, this meant the one and only device that was available to use: a five-

qubit prototype quantum processor made by IBM. So the grid for my Battleships game had five points, one for each qubit. To get my ships running on this real device, all I needed to do was use IBM's open-source Qiskit package to write my quantum program.

```
Press Enter to see the results for Player 1's ships...

Here is the percentage damage for ships that have been bombed.

 ?         ?
 |\       /|
 | \     / |
 |  \   /  |
 |   \ /   |
 |    ?    |
 |   / \   |
 |  /   \  |
 | /     \ |
 |/       \|
 ?        47%


Ships with 95% damage or more have been destroyed
```

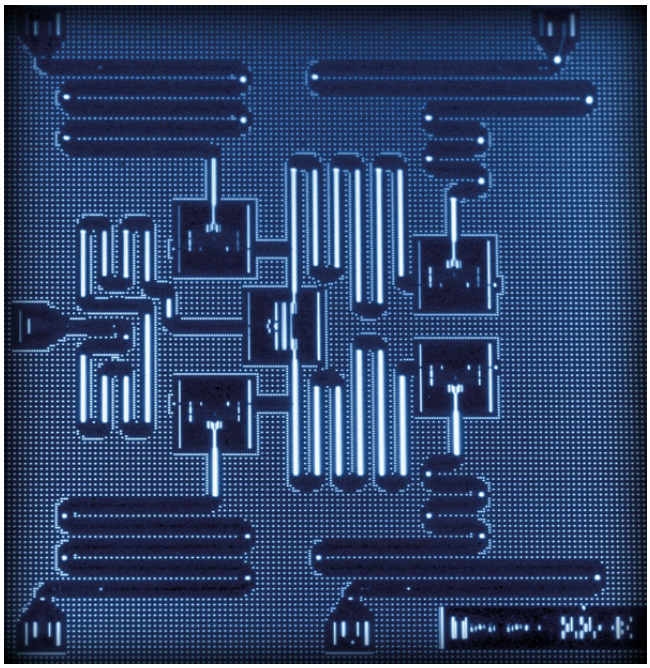

Figure 1(a) (top): Screenshot from Battleships with partial NOT gates, showing a ship that has been partly destroyed. Figure 1(b) (above): The five-qubit IBM device used for the game. The qubits are located within the five dark squares.

The game requires two players, each of whom must choose three of their five qubits to play the role of ships. For each qubit-ship chosen, we'll use the qubit state 0 to represent a ship that is intact, and 1 to represent one that is destroyed. The other player then has to try and sink these ships by turning each 0 into a 1. In terms of standard computing, this operation is known as a NOT gate. It is the simplest of the logic gates that underpin all digital computing. In our game, though, it simply plays the role of a successful attack.

Now let's add in some quantum. Qubits can, famously, exist in states other than just 0 or 1. They can also be in one of an infinite number of superpositions, some of which are weighted more towards 0, and some more towards 1. But if we actually measure the state of one of these superposition qubits, we force it to randomly choose between the two binary options. The weighting of the superposition determines the probabilities of each outcome.

Because quantum computers can access superposition states, they can perform partial versions of standard logic gates. We can, for example, make them do half of a NOT. Applying this to a qubit in a 0 state moves it into a superposition halfway between 0 and 1. If we run this operation many times, measuring the qubit each time to extract an output, we'll find that 0 and 1 come out with equal probability. The result, in the game, is a ship that is half destroyed.

If, instead, we did two of these half-NOT gates before making a measurement, something very different would happen. The first half-NOT would take the qubit state 0 and park it in a superposition between 0 and 1. Then the second would take this superposition and continue the journey. The result would be a qubit in state 1, and a ship that is destroyed.

This is how quantum superpositions and single-qubit rotations manifest themselves in the game: not as philosophical conundrums or arcane concepts reserved only for the initiated, but as partially damaged ships and not entirely effective weapons. We have taken the exotic elements of quantum programming and given them mundane jobs in a game. With their mysticism stripped away, it becomes easier to start thinking about what you might want to do with them.

This is the main goal of *Battleships with partial NOT gates*. Like *Spacewar!*, it aims to provide an example of programming for others to build upon. It is a game for people to look at and declare "I could do that." Because you could. In fact, you could do better.

Quantum awesomeness

Each run of *Battleships with partial NOT gates* uses only three of the five qubits on the device. To these qubits, we apply only one type of quantum operation. Clearly, we are not pushing the device to its limits. Hence, to find a better quantum heir for *Spacewar!*, we'll have to look elsewhere.

All the power of a quantum computer comes from its ability to explore the full "state space" for its qubits. For a single qubit, that means being able to achieve the states 0, 1 and all possible superpositions. For two qubits, the available space becomes more complex. The system has four basic states – 00, 01, 10 and 11 – and with these we can create more kinds of superposition states. As more qubits are added, the system becomes increasingly complex. For n qubits, there are 2^n basic states for us to put into superpositions: an exponential growth in the number of possibilities.

The vast majority of possible many-qubit states will exhibit some degree of entanglement – one of the signature aspects of quantum mechanics. Entanglement allows information to be stored non-locally across qubits, leading to effects that Albert Einstein referred to as "spooky action at a distance". Creating and manipulating entanglement in a controlled manner is notoriously difficult, and for the past few decades, creating and studying specific entangled states for a few qubits was easily enough to net you a PhD. But building a quantum computer is even more daunting. We need to make a device that can reliably create any entangled state we desire, for an arbitrarily large number of qubits.

One way to test whether a device makes the grade is by creating and running random quantum programs. A random quantum program does exactly what it says on the tin: it takes all the operations your quantum computer can do and throws them randomly into a program. This is typically run on a bunch of qubits that all start off in the 0 state: no superpositions and no entanglement. As it is run, superpositions are created and entanglement begins to build up. If you run it for long enough, you'll end up with a completely random example of one of the infinite possible states for your qubits, no matter how complex or entangled it might be. Then you just need to measure your qubits, do some statistics and prove that you got the state you expected given the program you ran. This is part of a test that Google hopes to run, which would serve as a proof-of-principle that quantum computers can do what would be practically impossible for normal ones.

Let's make this into a game. Suppose we have an opponent who creates a small random quantum program that involves randomly chosen pairs of qubits getting entangled in a random way. The program is then run, and we analyse the results. Our aim is to work out what our opponent did, and to add extra lines to the program that will undo it. In its basic concept, this game is a bit like *Tetris*. You, the player, must battle against the forces of chaos. You take whatever randomness the game throws at you and try to undo its effects as best you can. If you are good, you will be able to keep order for a long time. If you are inept, you will essentially become an extra source of randomness yourself, and the game will quickly become unplayable.

The incompetence of this game's players, and the random quantum programs the opponent creates, would require the quantum computer to create and manipulate complex entangled states almost constantly. It would provide a real test for the device, and it could even be used as part of experiments that prove the power of quantum computation. We'll give the game a name that reflects this: *Quantum Awesomeness*.

In the near-term, our experience of playing *Quantum Awesomeness* will be dominated by another, much less welcome effect. Qubits inevitably interact with their environment, and the operations we perform are never quite perfect. As these errors build up over long quantum programs, the results we get from a device will strongly deviate from the results we want and expect. Eventually, the output from each qubit will be just a coin flip, unrelated to any other qubit or the program that was run. Any complex superposition states will have long since decohered away.

There is, however, a silver lining to this problem, which is that playing *Quantum Awesomeness* could give us a feel for how noisy a device is. Once we have seen how many rounds can be played before the game becomes an exercise in frustration (think of playing *Tetris* with faulty controls that jam or misbehave more and more as each game goes on, and a flickering screen to compound the problem),

we will get a sense of how long our quantum programs can become while still producing good results.

Quantum Awesomeness also gives us another way to compare different devices. In quantum computers, entanglement is created via operations that interact with pairs of qubits. But not all qubit pairs can be interacted with directly. Each device will have a connectivity graph that lists all the pairs for which a particular two-qubit operation can be performed. The better connected a device is, the more flexible and adaptable it will be in creating quantum programs, and the faster it will be able to create complex superposition states.

Within *Quantum Awesomeness*, the connectivity graph becomes the board on which the game is played. The better connected the device, the more moves that both the opponent and player have at their disposal. The very properties that make a quantum processor more useful will also make its version of *Quantum Awesomeness* more engaging to play.

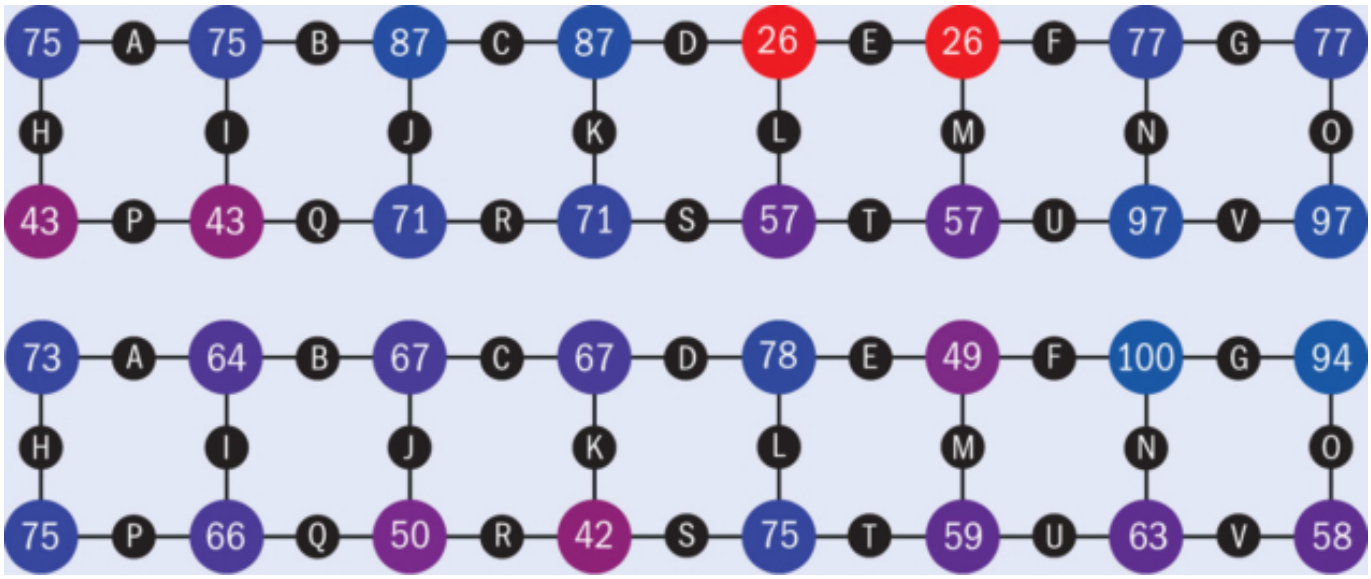

Figure 2: Grids from games of Quantum Awesomeness played on (*a*: top rows) an error-free simulation of IBM's 16-bit device and (*b*: bottom rows) the real physical system.

Figure 2 shows a *Quantum Awesomeness* board for IBM's publicly available 16-qubit device, which has a ladder-like connectivity graph. In each image, the coloured circles denote qubits. The numbers inside the circles give a measure of how entangled each qubit is, which is calculated from the results of measurements. The opponent entangles randomly chosen pairs of qubits. Using the fact that the numbers for the two qubits in each pair should be equal, the player's job is to work out which pairs were entangled.

In a hypothetical perfect quantum computer with no noise, this job is not difficult. This can be seen in the puzzle in figure 2*a* (top two rows), which was produced by an error-free simulation of the 16-qubit device. The two qubits in each pair have numbers that are exactly equal, making it easy to pick them out. The solution here is pairs A, C, E, G, P, R, T and V.

Things get trickier when we run the game on the real device. Results from one run are shown in figure 2*b* (bottom two rows). The presence of noise means that our measure of entanglement is not completely accurate, and so the numbers for the two qubits in each pair can differ. This can cause ambiguities that take a little more effort to resolve. For example, should the 63 to the lower right be paired with the neighbouring 59? Or with the 58? Although the 59 is closer in value to the 63, when we look at the neighbours of these numbers, we can see that it is actually pair V that is correct. The solution here is pairs C, G, H, I, L, M, R and V.

The first quantum hackers

The development of *Spacewar!* spurred the development of an emerging hacker culture (in the original, positive sense of the word). The game's code was freely shared, which meant that others were able to learn from it and adapt it. New features were added, variants were made, and it was (and still is) ported to a variety of different systems.

The nascent field of quantum programming has already begun to head in this same fruitful direction. If you want to program a job for a real quantum device, your choices are to use IBM's Qiskit, Rigetti's Forest or ProjectQ from ETH Zurich. All are open-source projects that encourage contribution. The same is true for the software that they run. Whether it be games, or scientific studies that lead to published papers, a lot of the source code is online and well documented – ready for newcomers to learn from, or to adapt and use themselves.

Until a few years ago, experimental quantum computing was something that you could only do if you worked in one of the right labs. Even theorists working in the same field had little access to it. Now, thanks to devices put online by IBM and Rigetti, using real quantum hardware is something that's accessible to all. You can run experiments while sitting in your pyjamas. You can try out a new idea without needing to write a grant application or try to convince venture capitalists that it will earn them a tonne of money.

You can even make games.

James Wootton is a quantum computing researcher who has just moved from the University of Basel, Switzerland, to the quantum computing group at IBM Research Zurich