Abdullah Assaf

Richard Noh

Emily Padilla

Quantum UNO (QUNO)

General Structure and Introduction

In our project, we aimed to create a quantum version of the popular card game UNO. Like the original game, we have a discard pile, a deck of cards, and players each holding a set of private cards. Players take turns playing a card, which must match the top of the discard pile in its type (like numbers) or color (like Red). The player who manages to get rid of all of their cards wins the game.

In our quantum version of the game, each card is represented by Card objects, each containing a 7-qubit quantum circuit, which stores the card's "true" state. We store 7 qubits because we require 2 qubits to represent the color (red, blue, yellow, and green), and 4 qubits to represent the type (9 numbers, 1 "Add Phase" type, and 6 "Entangle" types). The last qubit is the output qubit for use in quantum algorithms. Whenever a player plays a card, this quantum circuit is measured to result in a single 6-bit quantity, which represents the measured type and color. The exact details of the types and colors are in the Color and Type enums of card.py. Additionally, the deck always shows the color of the next card, which is important for the "interference" part of this game.

Since doing a full GUI and multiplayer is incredibly complex and time consuming, our game runs on a console window, which starts by simply running the command:

```
python -m quno.py
```

Assuming that qiskit is installed in the virtual environment, the program begins at the bottom of quno.py, introduces the player, and starts the game. We represent the overall state of the game in a Game object, which stores all of the Player objects (representing each player), and the Deck object that creates all of the Card objects. Each Player object primarily stores an array of Card objects, which represents that player's hand.

For each player's turn, the console will be cleared and prints out the state of the game. The player simply enters in the index of the card that they want to play, or "d" for pulling a card from the deck.

Superposition, Our "Wild" Cards, and Measurement

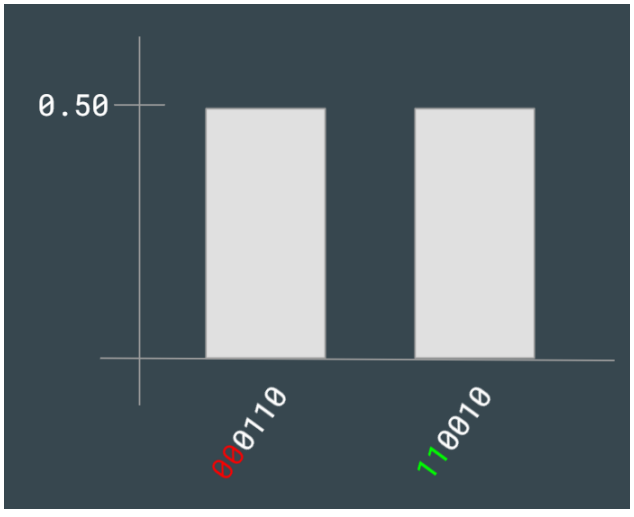
Unlike classic UNO, we do not have true wild cards but rather "superposition" cards. These are cards with two types and colors, for example, a Red 7 / Green 3 card. The player can

play this card as long as one of the types matches the type of the top of the discard pile or one of the colors matches the color. Whenever a superposition card is played, the underlying superposition circuit is measured, and the resulting discard pile card is either one of the superposition states (50/50 chance).

Initially, we wanted to systematically create the superposition quantum state with a series of gates, then simply measure all of the qubits to reach either state. However, programming this would have been a technical nightmare and may deserve a full research paper dedicated to its development. To illustrate the difficulty, let us go back to the Red 7 / Green 3 superposition card example. The Red 7 state is encoded in binary as 00 0110, while the Green 3 state is encoded as 11 0010 (the binary for all colors and types is in the code). We initially wanted to create the following quantum state:

$$\frac{|000110\rangle + |110010\rangle}{\sqrt{2}}$$

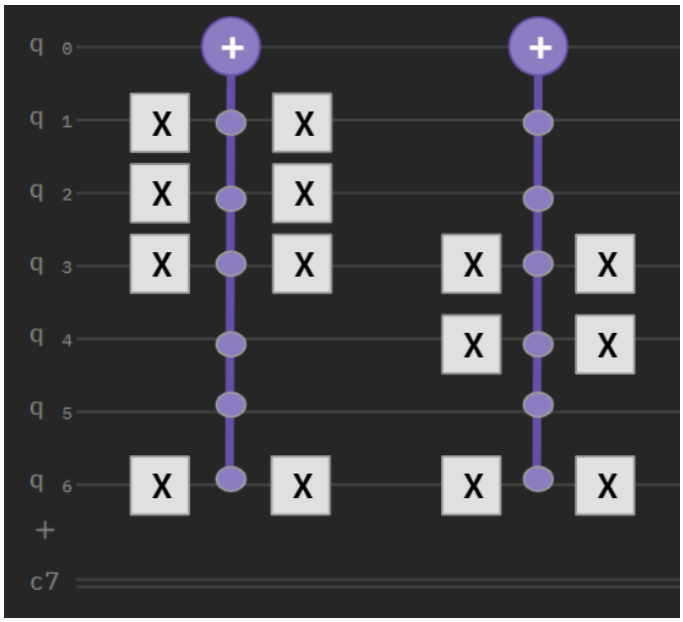
This state is nonseparable, requiring the heavy use of controlled-X gates and a lot of code. Fundamentally, the only thing we wanted to achieve after measuring the quantum circuit is a histogram that looks like:



We had the realization that this sort of result looks incredibly similar to the application of Grover's algorithm for M=2 solutions for a General Boolean Search problem. In this case, our boolean function, **f(x)**, was the function that gave a 1 whenever **x** was equal to the binary representation of the Red 7 or Green 3 state:

$$f(x) = (000110) \text{ or } (110010)$$

In the card's quantum circuit, we can represent this with two recursive Toffoli gates and a series of NOT gates which masks out the 0 bits in each solution:



After this oracle goes through Grover's algorithm, our measurements of this quantum circuit result in a 50/50 chance of reaching either state: Red 7 or Green 3, just how we wanted it to be.

Programmatically, we make every card that a player owns go through Grover's algorithm because doing it otherwise would require more conditionals and code that would make our lives harder. However, as a consequence, each time a player plays a card, the game freezes for approximately 5 seconds to run the massive quantum circuit. Most of the slow down is primarily due to these 6 controlled qubit Toffoli gates, which require dozens of sub gates to function.

Entanglement, Our “Skip +1” Cards

Our way of demonstrating entanglement has the effect of behaving similarly to the classic UNO “skip” card while also being a “+1” card. One of the primary types of cards that a player can get is a “Make Entangled X/Y” card, where X and Y are two of the four colors. An example of such a card would be “Make Entangled Red/Blue”. To easily explain this card, let us work through a scenario.

Suppose that two players, Alice and Bob, are in a game of QUNO - Alice plays first, then Bob goes after. It is currently Alice’s turn, and Alice has a “Yellow: Make Entangled Red/Blue” card in her hand. For the sake of this example, let’s suppose that there is a Yellow card on top of the discard pile. When Alice plays this card, an “entangled” pair is created. One of the two cards is given to Alice, let’s say the Red card, and the Blue card is given to Bob. For Alice, she automatically “measures” her entangled card and places her Red card onto the discard pile - the exact type of this new Red card is random. Additionally, Alice will know that one of the cards that Bob has is Blue, which may be strategically helpful when both players have few cards.

For Bob, he receives an “Entangled” card and cannot know what kind of card it is. To play this Entangled card, he must waste a full turn to “measure” it. Programmatically, this is represented by a simple “isEntangled” boolean value. Once Bob measures the Entangled card, the card behaves like normal. It is important to note that Bob is not required to measure his Entangled card immediately after Alice gives her the card. The Entangled card acts as a dead weight in Bob’s hand until he spends a turn to reveal it.

While this demonstrates entanglement to the player, using real entanglement to incorporate this is rather difficult, and we were unsure if this project required us to use real

entanglement for this entanglement mechanic. Therefore, our code decides which player receives which card based on a measurement of the following Bell state:

$$\frac{|01\rangle + |10\rangle}{\sqrt{2}}$$

We create a small, 2-qubit quantum circuit where qubit 0 represents Alice's card, and 1 represents Bob's card. When we measure qubit 0 and read the classical bit 0, which represents the Card color X, Alice will know that Bob will have Card color Y. Similarly, when we measure qubit 0 and read a 1, Alice will know that Bob has Card color X.

Interference Demonstration

For demonstrating interference, we implemented a new special card called "Add Phase". As mentioned in the introduction, our Deck object always tells the player the next card's color. Programmatically, we represent this color as a 2-qubit quantum circuit. Each time a player plays an "Add Phase" card, the game will add a Rotation Y gate with phase pi/2 onto qubit 0, while qubit 1 remains untouched by the card. We use a different circuit than the Card object's 7-qubit circuit because rotation Y gates will mess with the result in Grover's algorithm.

For example, suppose that the next card's color will be Blue, which we encoded as 01 in binary. Once a player plays a single "Add Phase" card, and the Rotation Y gate is applied, there will be a superposition between 00 (red) and 01 (blue). If a player pulls a card from the deck now, the quantum circuit is measured, and the resulting card will be either red or blue. If two "Add Phase" cards were played, and only then a player pulled from the deck, the card is guaranteed to be red. If the next card was a superposition, like Red and Green, the game will represent the first color in a quantum circuit and reinterpret the second state color based on the

measurement. Suppose that a player plays a single Add Phase card on the superposition. The two possible cards will be a Red/Green or a Blue/Yellow, never a “Green/Yellow”.

Conclusion

Given our time constraints and other academic/work obligations, we believe that we did extraordinarily well on this project. If we had additional time, we would make this game more efficient by not sending normal cards into Grover’s algorithm, add more detailed UI details (like the number of cards each player has), and maybe even a rudimentary over-LAN multiplayer.