



4107056005 資工四 方郁婷

使用方法

直接點開網頁使用即可

› 文件 › 四上課程 › 量子 › 量子電路計算機

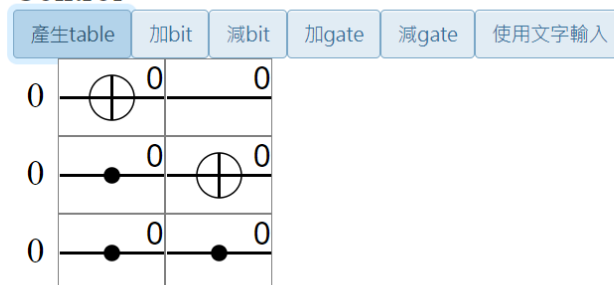
名稱	修改日期
 src	2021/11/14 下午 09:51
 index	2021/11/14 下午 09:49

使用介面

Table

input	output
000	000
001	011
010	010
011	101
100	100
101	111
110	110
111	001

Control



有兩種輸入方法

1. 藉由滑鼠點擊

可新增與減少 bit 與 gate

Control

產生table




加bit

減bit

加gate

減gate

使用文字輸入

	0	0	
	0		0
	0	0	

點擊 gate 可切換

Control

產生table

加bit

減bit

加gate

減gate

使用文字輸入

2. 使用文字輸入

為了方便助教批改新增了文字輸入的功能

Table

input	output

Control

產生table





加bit

減bit

加gate

減gate



使用文字輸入

以換行與空格隔開

Control

產生table

	2 3
	1 2
	2 3
	3 3

左方為輸入框

0 代表 0 control

1 代表 1 control

2 代表not

3 代表不做運算

輸入

產生 table

Table

input	output
0000	0100
0001	0101
0010	0110
0011	0111
0100	1000
0101	1001
0110	1010
0111	1011
1000	1100
1001	1101
1010	1110
1011	1111
1100	0000
1101	0001
1110	0010
1111	0011

Control

產生table




加bit

減bit

加gate

減gate

使用文字輸入

0		0	0	
0		0		1
0		0	0	0
0		0	0	0

額外加分(Bonus)

1. 視覺化界面(如上述)
2. 支援 Support Negative CNOT gate

Table

input	output
00	01
01	11
10	10
11	00

Control



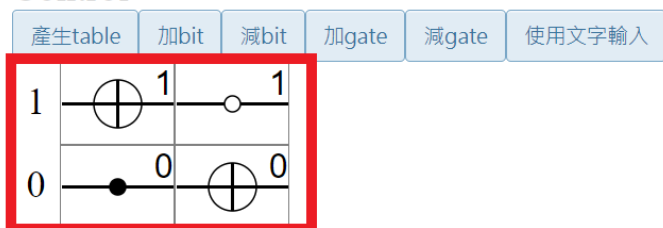
3. 支援 step by step 計算

點選 table 即可顯示

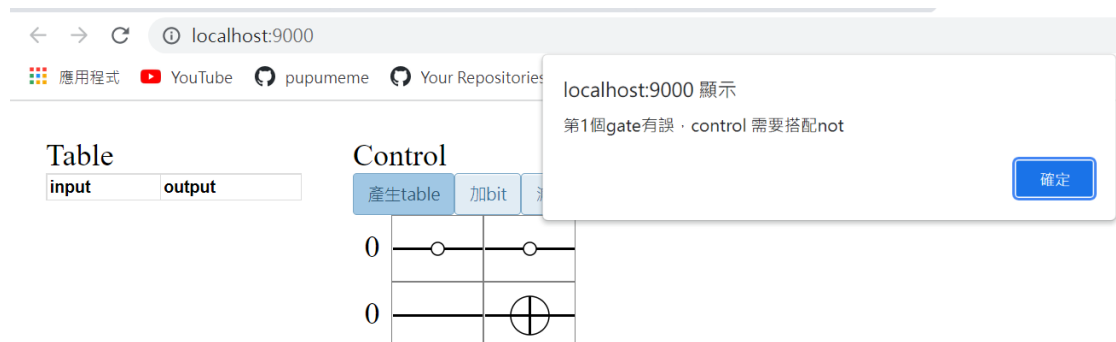
Table

input	output
00	01
01	11
10	10
11	00

Control



4. 防呆



localhost:9000 顯示

第1個gate有誤，有control時，not只能有一個

Table

input output

Control

產生table 加bit

localhost:9000 顯示

gate 的長度不一，請重填

Control

產生table

2 2

2 3

1

0

0

1 代表 1 control

2 代表not

3 代表不做運算

輸入

就算格式很亂照樣可以輸入

Control

產生table

1 2

2 3

1 1

左方為輸入框

0 代表 0 control

1 代表 1 control

2 代表not

3 代表不做運算

輸入

因為我使用了正規表達式去拆解

```
let newArr = text.split('\n').filter(function (obj) {
  return obj.trim() !== '';
}).map(function (obj) {
  return obj.trim().split(/\s+/).flatMap((v) => parseInt(v))
});
```

會自動轉為正確格式

Control

產生table

1 2

2 3

1 1

左方為輸入框

0 代表 0 control

1 代表 1 control

2 代表not

3 代表不做運算

輸入

5. Toggle 二進位、十進位

Toggle 完後要再點一次產生 table

Table

input	output
0	1
1	2
2	0
3	3

Control

產生table

加bit

減bit

加gate

減gate

轉十進位

使用文字輸入

0

0

0

0

1

1

6. 其他

老師說過一個 gate 只能有一個 not，我這邊是如果沒有 control 的話可支援多個單一的 not

Table

input	output
00	00
01	01
10	10
11	11

Control

產生table

加bit

減bit

加gate

減gate

使用文字輸入

0

1

0

1

0

0

1

0

1

0

實作流程

使用 ReactJS 前端框架

總程式碼架構

```
1 > import React, { useEffect, useState } from 'react'; ...
4
5 > function GateSVG({ id }) { ...
54 }
55 > function GateUnit({ id, row_pos, col_pos, setGate, bit_label }) { ...
64 }
65 > function RowGates({ row_pos, row_arr, setGate, row_result_bit }) { ...
84 }
85 > function BitUnit({ id }) { ...
91 }
92
93 > function AllGates() { ...
352 }
353 > function TextInput(...
449 }
450 //使用ReactDOM.render把剛建立的物件element插入目標DOM中
451 ReactDOM.render([
452   <AllGates />,
453   // <TextInput />,
454   document.getElementById('root')
455 ],
```

AllGates

Table

input	output
00	00
01	01
10	10
11	11

Control

產生table	加bit	減bit	加gate	減gate	使用文字輸入
0	\oplus 1	\oplus 0	\oplus 1	\oplus 0	
0	\oplus 1	\oplus 0	\oplus 1	\oplus 0	

TextInput

2 2 2 2
2 2 2 2

左方為輸入框

0 代表 0 control

1 代表 1 control

2 代表not

3 代表不做運算

輸入

RowGates

Control



GateUnit、GateSVG

Control



BitUnit

Control



其中 AllGates 是控制中心，掌控所有變數的設置

```
function AllGates() {  
  const [all_gates_arr, setAllGates] = useState(...  
);  
  const [all_bits_arr, setAllBits] = useState(...  
);  
  const [result_bit, setResultBits] = useState(...  
);  
  const [row_num, setRowNum] = useState(2);  
  const [col_num, setColNum] = useState(3);  
  const [table, setTable] = useState([]);  
  
  const [displayText, setDisplayText] = useState(false);  
}
```

主計算功能

在計算一個 col 時，會記得 not 的位置和 control 是否觸發

```
//新的result arr
let newArr = Array.from({ length: row_num }, _ => new Array(col_num).fill(-1));
// console.log(newArr);
for (var j = 0; j < col_num; ++j) {
  control_not = true;
  not_bit_pos = [];

  for (var i = 0; i < row_num; ++i) {
    // console.log(bit_arr[i], all_gates_arr[i][j]);
    if (all_gates_arr[i][j] == 3) { //line
      continue; //略過
    }
    if (all_gates_arr[i][j] == 2) { //not
      not_bit_pos.push(i); //記住是哪個bit要not
    }
    else if (bit_arr[i] != all_gates_arr[i][j]) { //不符合control
      control_not = false;
      break
    }
  }
}
```

因為我有支援當沒有 control 時可多個 not 運算

所以我的 not 是以陣列來裝，並且以陣列的長度判斷現在有幾的

not，之後再做運算，最後的 for 是用來更新 result bit 陣列

```
}
if (control_not && not_bit_pos.length == 1) {
  bit_arr[not_bit_pos[0]] = bit_arr[not_bit_pos[0]] == 0 ? 1 : 0
}
else if (not_bit_pos.length > 1) {
  for (let n = 0; n < not_bit_pos.length; ++n) {
    bit_arr[not_bit_pos[n]] = bit_arr[not_bit_pos[n]] == 0 ? 1 : 0
  }
}
for (let row_i = 0; row_i < row_num; row_i++) {
  newArr[row_i][j] = bit_arr[row_i];
}
```

程式原碼在 `src/index.js`

`src/bundle.js` 是編譯過的程式碼