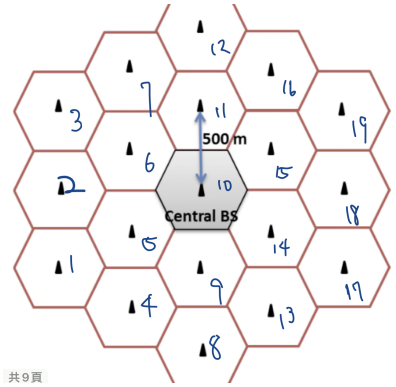
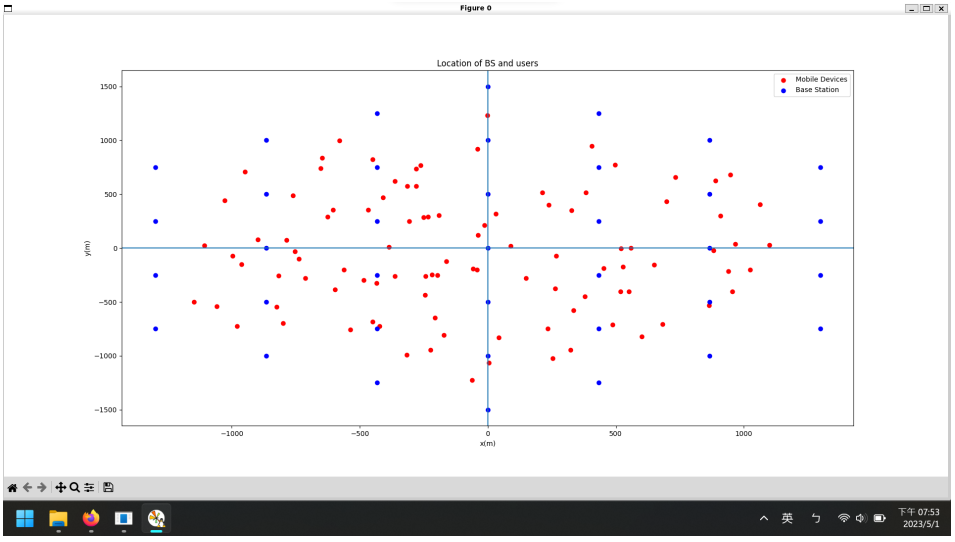
1.

我根據上次hw2 code的安排各cell中心的順序，來做這次的cell id的安排。由下往上，由左往右。



2.

各MS(紅點)初始位置和各BS的位置(藍點)如下圖，特別要提的是由於我程式的設計，在題目提到的19個cells外，外頭還有一圈BS。不過完全不影響MS初始位置的觀察，也有座標軸可以對照。



設計理念:在初始化的過程中，第一個想到的是，直接在19個cell構成的範圍撒下一百個點有些吃力，因為邊界的限制會變得非常複雜。恰巧上次作業中有寫出在一個六角形cell裡撒下幾個MS的code。所以我的想法是生成19個亂數，a,b,c…。再將它們全部相加當分母D，第一個cell的MS數即為100*a/D，再用我上次寫的那個code撒MS。因為乘除會有小數的問題，怕最後相加不是一百，所以用此方法產生18個數後，最後一個cell的數字用100去剪掉前面18個數的和。然而這樣的做法我發現因為python 的int強制轉型是無條件捨去，所以最後一個數字通常會變很大 (前面18個小數無條件捨去的累積)。所以我最終的做法用100*a/D這種方法

產生19個數後(用one_hundred_MS這個list裝)，不足100的(code中的remaining)再用上述方法分配一次，到0為止。

```python
def gen_19_num():
    one_hundred_MS = []
    for i in range(19):
        one_hundred_MS.append(random.randrange(10, 100))
    s = sum(one_hundred_MS)

    for i in range(19):
        one_hundred_MS[i] = int(100 * one_hundred_MS[i] / s)

    remaining = 100 - sum(one_hundred_MS)
    integer = 0
    while remaining != 0:
        adding = random.randrange(0, 2)
        if adding == 1:
            one_hundred_MS[integer] += 1
            remaining -= 1
        integer += 1
        if integer == 19:
            integer = 0

    return one_hundred_MS
```

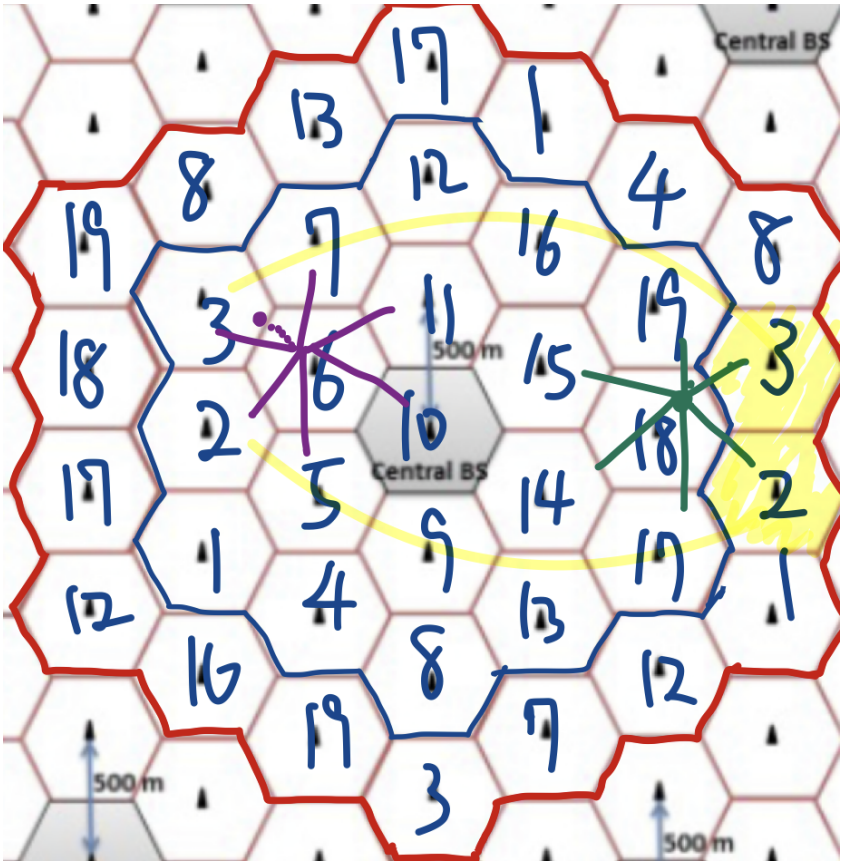
3.

Time(s)	Source cel	Destination c
0.3	9	4
2	16	1
2.1	2	18
2.1	13	12
2.6	9	10
3.6	9	5
9.2	12	13
9.3	9	10
9.5	4	9
9.5	19	3
9.6	15	14
9.7	11	7
9.9	4	9
10	1	2
10.6	15	11
10.8	1	16
11.1	1	12
11.4	10	11
12.2	10	9
12.8	19	15
12.9	10	9
13	11	16
13.1	19	3
13.8	19	16
13.9	19	3
14	19	16
14.3	7	11
14.6	3	18
15.4	9	4
15.5	13	12
16.8	2	1
16.8	9	10

我將結果記錄成CSV，程式執行完畢會自動匯出。

Handoff的判斷標準拿兩個不同位置的MS來舉例。先說明一下，我這次設計的code。裡面19個cell是題目中的，外面則是我為了方便計算SINR而加的。像是綠色MS地理位置位在第18個cell，而它也是連接第18個cell的BS。所以下一個時間單位(我以每0.1秒當作一個時間單位)，就要做一次移動，並重新計算綠色MS連接第18個BS的SINR"以及"綠色MS和第18個BS周圍6個cell(19, 3, 2, 17, 14, 15)的BS連接的SINR。而實際上第18個cell旁邊第2和3的cell是我多加的，好方便計算相對位置，但第2和3的BS(塗黃色處)受到的干擾和裡面的第2和3的BS是相同的(黃線指向的地方)。

而紫色MS雖然地理位置在第3個cell但或許因為第3個BS受到的干擾太大，紫色MS實際上是連接第6個cell的BS。所以下次移動後就要以第6個cell的BS為中心和周圍cell(7, 11, 10, 5, 2, 3)算SINR，當然也包括和第6個cell的BS算SINR。我原本是不用地理位置去作為中心是因為要判斷每個MS的位置和附近的BS會耗掉比較多計算時間(因為不固定)，而用每個BS和它們連接的MS算SINR再用BS的周圍6個算會比較有規則(我設計的BS是一個class，其中一個attribute就是連接到的MS名單)。但後來發現可能會有一些地理位置和連接到的BS位置有一段距離，只判斷BS的周圍cell有些偏頗。然而後來又想到這次作業的，cell的距離不遠，也蠻常更新一次，所以上述問題也不會到太嚴重。



而定量的判斷handoff便是，偵測到的新SINR比原本的大3dB便做轉換。我看講義上是寫90~120 dBm，但作業的cell距離近，加上我實測SINR都相差個位數到十位數(dB)，所以我認為要做一些調整，剛好3dB對應到2倍，是我認為比較合理的數字。

```
if (outer_SINR[neighbor] - local_SINR > 3):  # Threshold
```

4.

根據我做的結果handoff的結果，總次數大多落在1800~2400內(我code執行完畢會在終端機輸出。

5.

改善:這次作業算是第一次從頭刻一個這種類型的模型，還蠻多可以再改善的地方，像是用地理位置當中心和周遭BS算SINR會是比較好的選擇。另外，多加一圈外圍的cell，在一開始算SINR時會稍微方便一些些，但如果真的和對面的cell連上，地理位置卻還留在原地，接下的計算會非常繁複。好比下圖，如果綠色MS和右邊的cell 2連上後，我的code實際上是把它連接到19個cell裡的那個第二個cell(黃線所指處)，就變成綠色地理位置在右邊，但連接到的MS在左邊。我後來加上許多繁雜的限制才解決(有一個看起來完全是用人眼一個一個列舉的函式 `def in_to_out` 便是這樣來的)，反而比一開始我覺得用地理位置判斷應該複雜許多。還有許多可以改善的地方，我都記在註解，給以後的自己改善的空間!