Lab 6. Planning wireless Wi-fi networks

Department of Computer Science

Student: Patrushev Boris 19213

Advisor: Ruslan V. Akhpashev

Date: 29/04/2021

1.Calculating the size of the coverage radius of Wi-Fi networks

First, let's import some libraries that will be useful to us in this lab:

- numpy - for random parameters and working with arrays
- math - for working with log10, sqrt
- matplotlib - for working with plots, heatmap, colormap
- pandas - for working with data frames
- sklearn - for working with LinearRegression(MathStat's stuff)

In [1]:

```
from numpy import random
import math
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

Choosing random parameters from the table for Access Point(WiFi) and User Equipment

In [2]:

```python
WiFi = {
    "Label" : "Wifi",
    "Height" : round(random.uniform(0, 3), 1), #meters
    "TxPower": random.choice([100, 200, 300, 600]), #mWatt
    "AntennaGain": random.randint(10, 22), #dBi
    "NoiseFigure": round(random.uniform(3.5, 5), 1), #dB
    "Bandwidth" : random.choice([5, 10, 15, 20]), #MHz
    "ReqSINR" : random.randint((-5), 30), #dB
                ##Common parameters
    "CarriereFrequency" :  random.choice([2.4, 5]), #GHz
    "BuildingPenetration" : random.randint(8, 26), #dBi
    "InterferenceMargin" : random.randint(3, 6), #dB
}


UE = {
    "Label": "UE",
    "Height" : round(random.uniform(0, 3), 1), #meters
    "TxPower" : random.choice([100, 200]), #mWatt
    "AntennaGain" : 0, #dBi
    "NoiseFigure" :  round(random.uniform(6.5, 8), 1),  #dB
    "Bandwidth" : random.choice([5, 10, 15, 20]), #MHz
    "ReqSINR" : random.randint((-5), 30), #dB
                ##Common parameters
    "CarriereFrequency" : WiFi["CarriereFrequency"], #GHz
    "BuildingPenetration": WiFi["BuildingPenetration"], #dBi
    "InterferenceMargin": WiFi["InterferenceMargin"] #dB
}

for key, value in (WiFi.items()):
    print(key, ' : ', value)
print('_______________________________')
for key, value in (UE.items()):
    print(key, ' : ', value)
```

```
Label  :  Wifi
Height  :  1.7
TxPower  :  100
AntennaGain  :  20
NoiseFigure  :  4.3
Bandwidth  :  15
ReqSINR  :  9
CarriereFrequency  :  2.4
```

```
BuildingPenetration  :  12
InterferenceMargin  :  3
________________________________
Label  :  UE
Height  :  0.3
TxPower  :  200
AntennaGain  :  0
NoiseFigure  :  7.0
Bandwidth  :  20
ReqSINR  :  17
CarriereFrequency  :  2.4
BuildingPenetration  :  12
InterferenceMargin  :  3
```

Description of the process for calculating the maximum allowable losses (formulas, you can use pieces of code);

1. To calculate everything we need to convert the TxPower from *mWatts* to *dBm*
2. Thermal noise is a noise that is a result of the thermal agitation of electrons. The thermal noise power depends of the bandwidth and temperature of the surroundings.
3. Free-space path loss(FSPL)

In [3]:

```python
def dBm(mw):
    return 10 * math.log10(mw)

def thermalNoise(bw):
    return - 174 + 10 *  math.log10(bw*10**6)

def FSPL(frequency, distance):
    return 22.7 + 26 * math.log10(frequency) + 36.7 *  math.log10(distance)
```

1. Convert the *TxPower* from [mWatts] to [dBm]
2. Finding *RxSens* of UE and WiFi by sum *Bandwidth*, *ReqSINR* and *NoiseFigure* [dBm]
3. Finding *Maximum Allowed Path Losses(MAPL)* [dB]
4. Findong *MaxDistance* of signal [m]

In [4]:

```python
for equipment in [WiFi, UE]:
    equipment["TxPower"] = dBm(equipment["TxPower"])
    equipment["RxSens"] = thermalNoise(equipment["Bandwidth"]) + equipment["ReqSINR"] + equipment["NoiseFigure"]

WiFi["MAPL"] = UE["TxPower"] + UE["AntennaGain"] - WiFi["RxSens"] - UE["BuildingPenetration"] - UE["InterferenceMargin"]
```

```
UE["MAPL"] = WiFi["TxPower"] + WiFi["AntennaGain"] - UE["RxSens"] - WiFi["BuildingPenetration"] - WiFi["InterferenceMargin"]
#WiFi["RxPower"] = WiFi["TxPower"] + WiFi["AntennaGain"] - WiFi["BuildingPenetration"] - FSPL(WiFi["CarriereFrequency"], )

for equipment in [WiFi, UE]:
    equipment["MaxDistance"] = 10**((equipment["MAPL"] - 26 * math.log10(equipment["CarriereFrequency"]) - 22.7) /36)

for key, value in (WiFi.items()):
    print(key, ' : ', value)
print('_______________________________')
for key, value in (UE.items()):
    print(key, ' : ', value)
```

```
Label  :  Wifi
Height  :  1.7
TxPower  :  20.0
AntennaGain  :  20
NoiseFigure  :  4.3
Bandwidth  :  15
ReqSINR  :  9
CarriereFrequency  :  2.4
BuildingPenetration  :  12
InterferenceMargin  :  3
RxSens  :  -88.93908740944319
MAPL  :  96.949387366083
MaxDistance  :  61.36008809897475
_______________________________
Label  :  UE
Height  :  0.3
TxPower  :  23.010299956639813
AntennaGain  :  0
NoiseFigure  :  7.0
Bandwidth  :  20
ReqSINR  :  17
CarriereFrequency  :  2.4
BuildingPenetration  :  12
InterferenceMargin  :  3
RxSens  :  -76.98970004336019
MAPL  :  101.98970004336019
MaxDistance  :  84.70219403350482
```

Calculation of the maximum radius for the generated values.

In [5]:

```
max_distance = min(UE["MaxDistance"], WiFi["MaxDistance"])
max_radius = math.sqrt((max_distance**2 - abs(UE["Height"] - WiFi["Height"])**2))
print(f'Maximum radius {max_radius} meters')
```

```
Maximum radius 61.34411472597793 meters
```

2. Visualization of the heatmap

Select and prepare a room layout for potential Wi-fi network planning

Also, with the help of this task, I learned that I live in a panel house type 111-90 :)

In order to draw the apartment in which I live, we need the following functions:

- We define some constants for AccessPoint, Walls and Doors
- add_wall, draw_wall & wall_intersect - for adding wall with coords in class, drawing and for calculating pen_los
- add_door, draw_door & door_intersect - for adding wall with coords in class, drawing and for calculating pen_los

In [6]:

```
WALL = -200
DOOR = -190
WIFI = 20

class Room:
    def __init__(self, width, length, unit):
        self.width = int(width/unit)
        self.length = int(length/unit)
        self.unit = unit
        self.plan = np.zeros((int(width/unit), int(length/unit)))
        self.walls = []
        self.doors = []

    def add_wall(self, point1, point2):
        point1 = self.meters_to_indexes(point1)
        point2 = self.meters_to_indexes(point2)
        self.walls.append((point1, point2))
        self.draw_wall(point1, point2)

    def add_door(self, point1, point2):
        point1 = self.meters_to_indexes(point1)
        point2 = self.meters_to_indexes(point2)
        self.doors.append((point1, point2))
        self.draw_door(point1, point2)

    def meters_to_indexes(self, point):
        x, y = point
        x = int(x / self.unit)
```

```python
        y = int(y / self.unit)
        return (x, y)

    def draw_wall(self, point1, point2):
        y1, x1 = point1
        y2, x2 = point2
        if (x1 - x2 == 0):
            y1, y2 = min(y1, y2), max(y1, y2)
            while(y1 < y2 and y1 < self.width):
                self.plan[y1][x1] = WALL
                y1 += 1
        if(y1 - y2 == 0):
            x1, x2 = min(x1, x2), max(x1, x2)
            while(x1 < x2 and x1 < self.length):
                self.plan[y1][x1] = WALL
                x1 += 1

    def draw_door(self, point1, point2):
        y1, x1 = point1
        y2, x2 = point2
        if (x1 - x2 == 0):
            y1, y2 = min(y1, y2), max(y1, y2)
            while(y1 < y2 and y1 < self.width):
                self.plan[y1][x1] = DOOR
                y1 += 1
        if(y1 - y2 == 0):
            x1, x2 = min(x1, x2), max(x1, x2)
            while(x1 < x2 and x1 < self.length):
                self.plan[y1][x1] = DOOR
                x1 += 1

    def add_router(self, point, router):
        point = self.meters_to_indexes(point)
        y,x = point
        self.plan[y][x] = WIFI;
        self.router_point = point
        self.router = router

#https://bryceboe.com/2006/10/23/line-segment-intersection-algorithm/
    def ccw(A,B,C):
        return (C[1]-A[1]) * (B[0]-A[0]) > (B[1]-A[1]) * (C[0]-A[0])
    def intersect(A,B,C,D):
        ccw = Room.ccw
        return ccw(A,C,D) != ccw(B,C,D) and ccw(A,B,C) != ccw(A,B,D)
```

```
#https://bryceboe.com/2006/10/23/line-segment-intersection-algorithm/

    def wall_intersect(self, point):
        count = 0
        y, x = point
        for point1, point2 in self.walls:
            if Room.intersect(point1, point2, point, self.router_point):
                count+=1
        return count

    def door_intersect(self, point):
        count = 0
        y, x = point
        for point1, point2 in self.doors:
            if Room.intersect(point1, point2, point, self.router_point):
                count+=1
        return count
```

I found a floor plan for a very similar three-room apartment on the Internet I set walls, doors and access point on the grid. And also calculate power of each cell

In [7]:

```
WIDTH = 9.5
LENGTH = 10
room = Room(WIDTH, LENGTH, 0.3)
#Walls:
room.add_wall((6.5, 0), (6.5, 4))
room.add_wall((2.5, 0), (2.5, 4))
room.add_wall((0, 5.5), (6.5, 5.5))
room.add_wall((6.5, 5.5), (6.5, 10))

#toilet&bath
room.add_wall((8.5, 5), (9.5, 5))
room.add_wall((8.5, 3), (8.5, 4.5))
room.add_wall((8.5, 5), (8.5, 5.7))

#Doors:
    #myroom
room.add_wall((6.5, 3), (7, 3))
room.add_door((7, 3), (8, 3))
room.add_wall((8, 3), (9.5, 3))
    #guest
room.add_wall((2.5, 4), (3.5, 4))
room.add_door((3.5, 4), (5.5, 4))
room.add_wall((5.5, 4), (6.5, 4))
    #bedroom
room.add_wall((6.5, 6), (7, 6))
room.add_door((7, 6), (8, 6))
room.add_wall((8, 6), (9.5, 6))
    #kitchen
room.add_wall((0, 4), (1, 4))
room.add_door((1, 4), (2, 4))
room.add_wall((2, 4), (3, 4))
    #toilet&bath
room.add_door((8.5, 4.5), (8.5, 5))
room.add_door((8.5, 5.7), (8.5, 6))
room.add_router((7, 1), WiFi)


for y in range(room.width):
    for x in range(room.length):
        if(room.plan[y, x] == WALL or room.plan[y, x] == WIFI  or room.plan[y, x] == DOOR):
            continue
```

```
        penetration_loses = room.wall_intersect((y,x))*32 + room.door_intersect((y,x))*8
        y1, x1 = room.router_point
        x1 = x1 * room.unit
        y1 = y1 * room.unit
        y2, x2 = y * room.unit, x * room.unit
        power = (room.router["TxPower"] + room.router["AntennaGain"] - penetration_loses
                 -FSPL(room.router["CarriereFrequency"], math.sqrt(abs((y2-y1)**2+(x2-x1)**2))))
        room.plan[y][x] = power
```

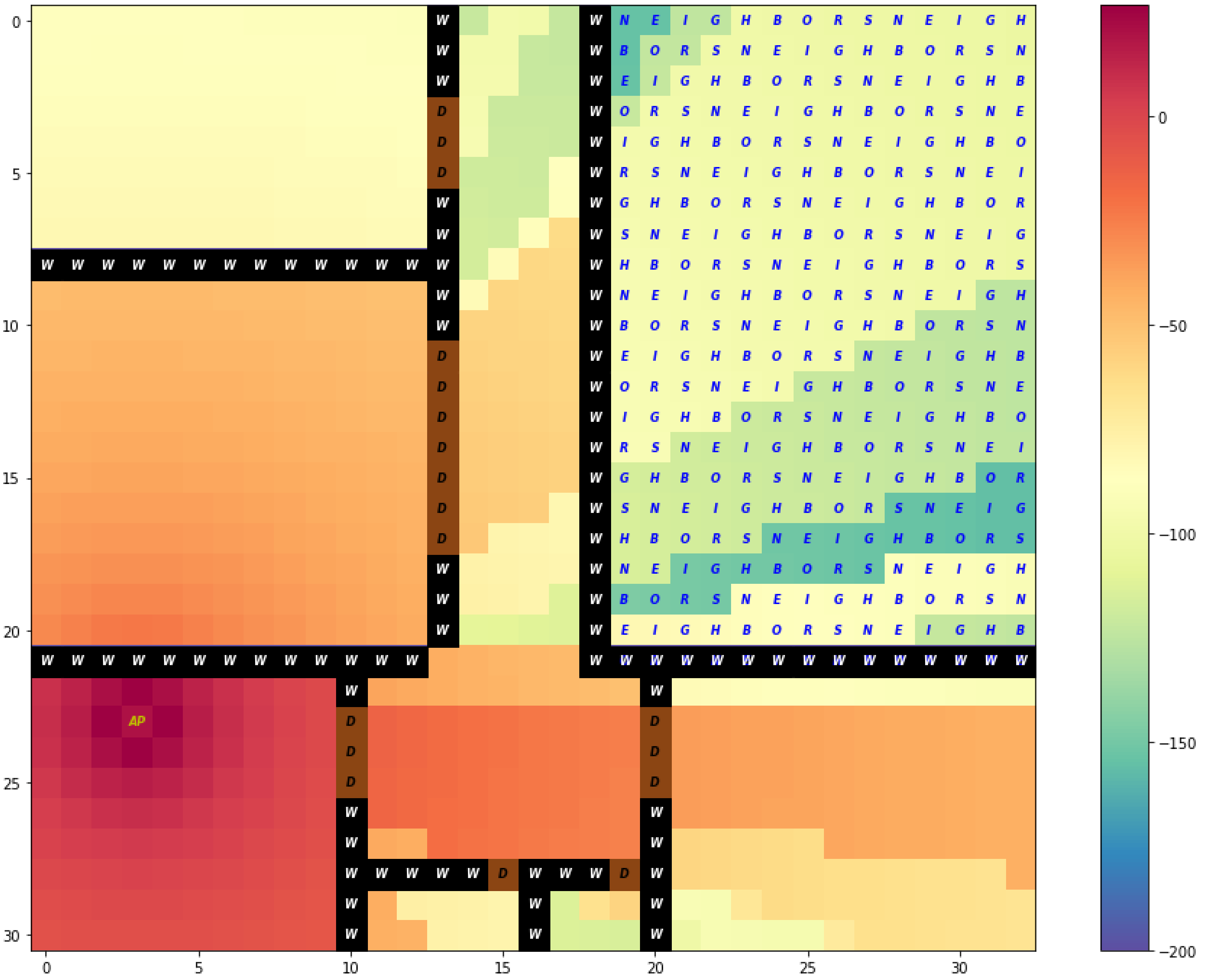
Rendering a heatmap

I render heatmap WiFi and also do some pretty stuff

In [8]:

```
def switch(value):
    if value == 1:
        return "N"
    if value == 2:
        return "E"
    if value == 3:
        return "I"
    if value == 4:
        return "G"
    if value == 5:
        return "H"
    if value == 6:
        return "B"
    if value == 7:
        return "O"
    if value == 8:
        return "R"
    if value == 0:
        return "S"

def change_color_for_spec():
    for y in range(room.width):
        for x in range(room.length):
            if (room.plan[y, x] == WALL):
                ax.scatter(x, y, s = 441, c='black', marker="s")
            if (room.plan[y, x] == DOOR):
                ax.scatter(x, y, s = 441, c='saddlebrown', marker="s")


#heatmap
```

```
fig = plt.figure(figsize=(16, 12))
ax = fig.add_subplot(111)

change_color_for_spec()

im = ax.imshow(room.plan, origin='upper', interpolation='None', cmap='Spectral_r')
fig.colorbar(im)

cnt_of_letter = 1
for y in range(room.plan.shape[0]):
    for x in range(room.plan.shape[1]):
        if((5.5 / room.unit < x < 10 / room.unit)&(-1/room.unit < y < 6.5/room.unit)):
            letter = switch(cnt_of_letter%9)
            cnt_of_letter+=1
            text = ax.text(x, y, letter, ha="center", va="center",
                           color="b", size="8", style='oblique', fontweight='bold')
        if(room.plan[y,x] == WIFI):
                ax.text(x, y, 'AP', ha="center", va="center",
                        color="y", size="8", style='oblique', fontweight='bold')
        if(room.plan[y,x] == DOOR):
                ax.text(x, y, 'D', ha="center", va="center",
                        color="black", size="8", style='oblique', fontweight='bold')
        if(room.plan[y,x] == WALL):
                ax.text(x, y, 'W', ha="center", va="center",
                        color="w", size="8", style='oblique', fontweight='bold')
```


AP
W
D
NEIGHBORS
0
5
10
15
20
25
30
−200
−150
−100
−50
0

3. Optimization of the radio signal propagation model

Collected experimental data(by "Wifi Analyzer") from different distances from the access point and recorded them in the room.csv

In [9]:

```
df_room = pd.read_csv("room.csv")
df_room
```

Out[9]:

	distance	dbi
0	0.1	-23
1	0.5	-35
2	0.6	-40
3	1.0	-40
4	1.5	-45
5	1.6	-52
6	2.0	-47
7	2.5	-51
8	3.1	-56
9	3.5	-56
10	4.2	-72
11	4.9	-63
12	5.0	-72
13	5.1	-72
14	7.0	-68
15	8.0	-72

Apply propagation function to distance column with sum of power parameters for optimization purposes

In [10]:

```
def FSPL(d):
```

```
    return 22.7 + 26 * math.log10(room.router["CarriereFrequency"]) + 36.7 * math.log10(d)
pen_loss_mean = 20
df_room.distance = room.router["TxPower"] + room.router["AntennaGain"] - df_room.distance.apply(FSPL) - pen_loss_mean
df_room
```

Out[10]:

	distance	dbi
0	24.114508	-23
1	-1.537691	-35
2	-4.443643	-40
3	-12.585492	-40
4	-19.048041	-45
5	-20.076696	-52
6	-23.633293	-47
7	-27.189891	-51
8	-30.618466	-56
9	-32.552790	-56
10	-35.458741	-72
11	-37.915688	-63
12	-38.237691	-72
13	-38.553318	-72
14	-43.600590	-68
15	-45.728895	-72

Divide the data into “attributes” and “labels”

And split 80% of the data to the training set while 20% of the data to test

In [11]:

```
X = df_room.distance.values.reshape(-1, 1)
y = df_room.dbi.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.01, random_state=11)
```

Train the algorithm:

In [12]:

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[12]: LinearRegression()

Get predicted values and compare it with actual

In [13]:

```
y_pred = regressor.predict(X)
df = pd.DataFrame({'Actual': y.flatten(), 'Predicted': y_pred.flatten()})
df
```

Out[13]:

	Actual	Predicted
0	-23	-16.619249
1	-35	-36.575890
2	-40	-38.836634
3	-40	-45.170748
4	-45	-50.198417
5	-52	-50.998679
6	-47	-53.765605
7	-51	-56.532532
8	-56	-59.199861
9	-56	-60.704706
10	-72	-62.965449
11	-63	-64.876880
12	-72	-65.127389
13	-72	-65.372937
14	-68	-69.299563

	Actual	Predicted
15	-72	-70.955320

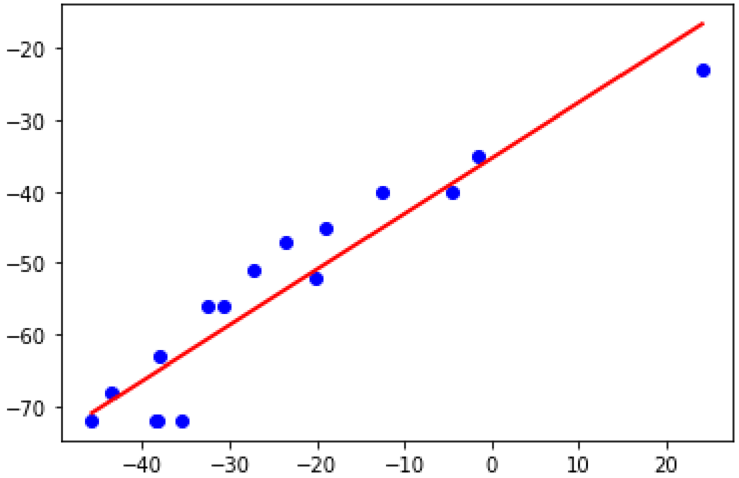
Print regressor coef and intercept. It's A and B, respectively

Show linear solving function and Mean squared error

In [14]:

```
print(regressor.coef_)
print(regressor.intercept_)

plt.scatter(X, y,  color='b')
plt.plot(X, y_pred, color='r', linewidth=2)
plt.show()

print('Mean Squared Error:', metrics.mean_squared_error(y, y_pred))
```

```
[[0.77796999]]
[-35.37961258]
```




```
Mean Squared Error: 24.196630958069026
```

Rendering optimized heatmap

In [15]:

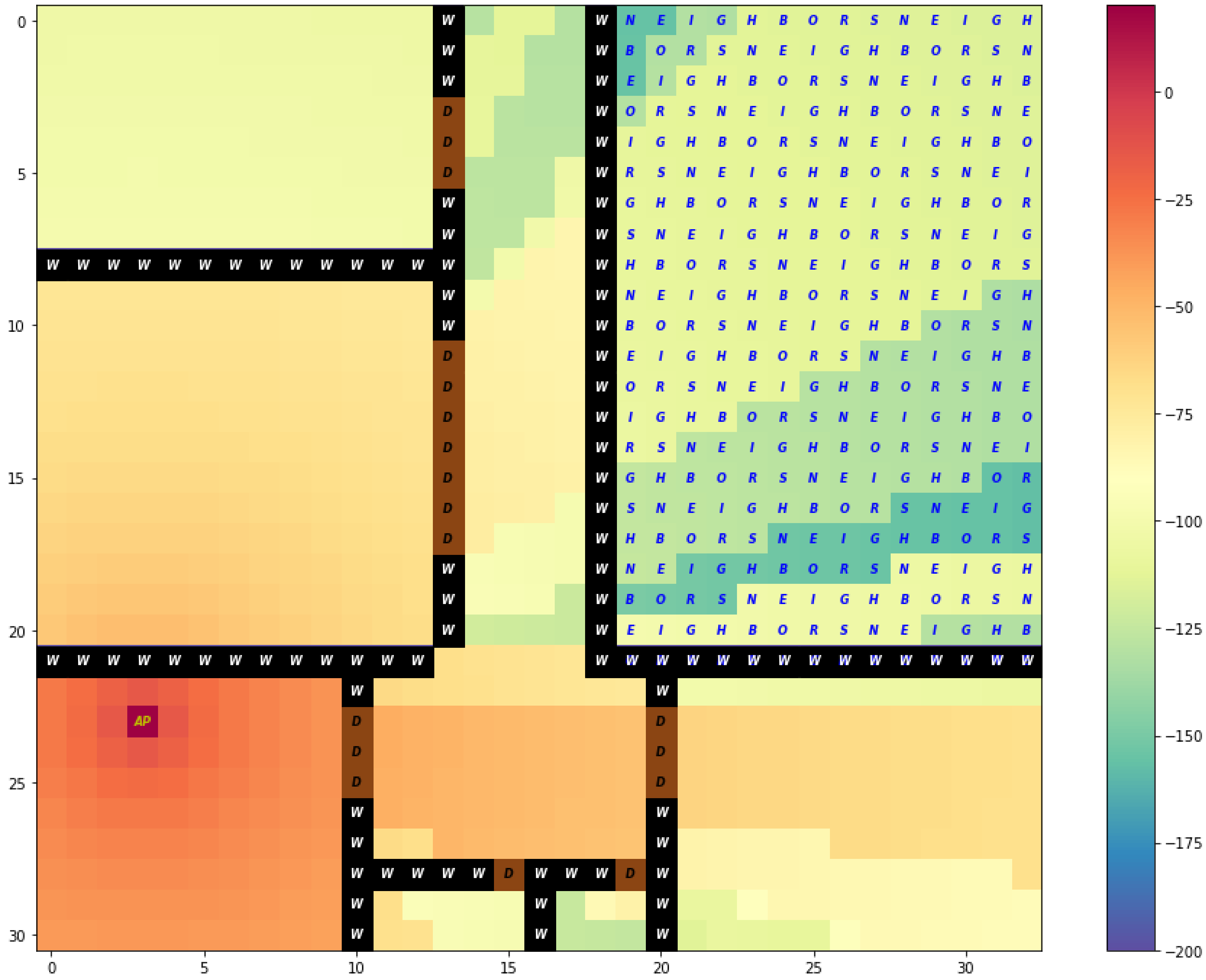
```
for y in range(room.width):
    for x in range(room.length):
        if(room.plan[y, x] == WALL or room.plan[y, x] == WIFI  or room.plan[y, x] == DOOR):
            continue
```

```python
        penetration_loses = room.wall_intersect((y,x))*32 + room.door_intersect((y,x))*8
        y1, x1 = room.router_point
        x1 = x1 * room.unit
        y1 = y1 * room.unit
        y2, x2 = y * room.unit, x * room.unit
        power = (room.router["TxPower"] + room.router["AntennaGain"] - penetration_loses
                 -FSPL(math.sqrt(abs((y2-y1)**2+(x2-x1)**2))))
        room.plan[y][x] = regressor.coef_*power+ regressor.intercept_
```

In [16]:

```python
def switch(value):
    if value == 1:
        return "N"
    if value == 2:
        return "E"
    if value == 3:
        return "I"
    if value == 4:
        return "G"
    if value == 5:
        return "H"
    if value == 6:
        return "B"
    if value == 7:
        return "O"
    if value == 8:
        return "R"
    if value == 0:
        return "S"

def change_color_for_spec():
    for y in range(room.width):
        for x in range(room.length):
            if (room.plan[y, x] == WALL):
                ax.scatter(x, y, s = 441, c='black', marker="s")
            if (room.plan[y, x] == DOOR):
                ax.scatter(x, y, s = 441, c='saddlebrown', marker="s")


#heatmap
fig = plt.figure(figsize=(16, 12))
ax = fig.add_subplot(111)

change_color_for_spec()
```

```
im = ax.imshow(room.plan, origin='upper', interpolation='None', cmap='Spectral_r')
fig.colorbar(im)


cnt_of_letter = 1
for y in range(room.plan.shape[0]):
    for x in range(room.plan.shape[1]):
        if((5.5 / room.unit < x < 10 / room.unit)&(-1/room.unit < y < 6.5/room.unit)):
            letter = switch(cnt_of_letter%9)
            cnt_of_letter+=1
            text = ax.text(x, y, letter, ha="center", va="center",
                           color="b", size="8", style='oblique', fontweight='bold')
        if(room.plan[y,x] == WIFI):
                ax.text(x, y, 'AP', ha="center", va="center",
                        color="y", size="8", style='oblique', fontweight='bold')
        if(room.plan[y,x] == DOOR):
                ax.text(x, y, 'D', ha="center", va="center",
                        color="black", size="8", style='oblique', fontweight='bold')
        if(room.plan[y,x] == WALL):
                ax.text(x, y, 'W', ha="center", va="center",
                        color="w", size="8", style='oblique', fontweight='bold')
```



Conclusion: In this lab, I learned how to calculate the maximum coverage radius of the AccessPoint, build the layout of my apartment, build a heatmap and optimize values with real data. In addition, I learned the type of panel house, played around with the location of the access point in the apartment and refreshed my knowledge of mathematical statistics from the last semester)