University of Bourgogne Franche Comté

Radio Networks - RI53

Project-CS: Cell selection

Final Report – Team 3

Done by

Hovhannes STEPANYAN, Massilva BOUTIAB, Mandy-Ann OWUSU MANU, Md Aftab UDDIN

Supervisor: Mr. MABED and Mrs. BAALA

June 10, 2022

In the final part we used the algorithm presented and explained by my teammates in the last reports to create the simulation. We used the code provided by Mr. MABED and made significant changes to implement cell selection feature. Besides creating simulation and parts to interact with it we also separated and modeled the code as well as made few interface improvements.

Tools used:

- Visual Studio Code and WebStorm -> Code editor
- JavaScript -> Programming Language

Manual to run the cell selection simulation:

After selection a rectangle in the map put some antennas in it. Then you are supposed to click on “Cartography of power” button. It will show the field of the power for each antenna only after that you can click on “Cartography of handover” button. On the map it will show two colors: “Green” represents the region where there is a meaning in hand overing; in other words more than one antenna is available to connect and “Blue” show the area where mobile is connected and there is no possibility to select other antennas. After this you should click on “Path creation Button” which is the last button in the footer. After clicking, select a start point on the map, a mobile will appear at that point. Next by clicking on map create a path for the mobile to move. When you finish path creation click on the “Finish path creation” button in the footer. After that a “Start” button will appear in footer. Click on it if you want to start the simulation. You can pause or start over the simulation with the last buttons in the footer.

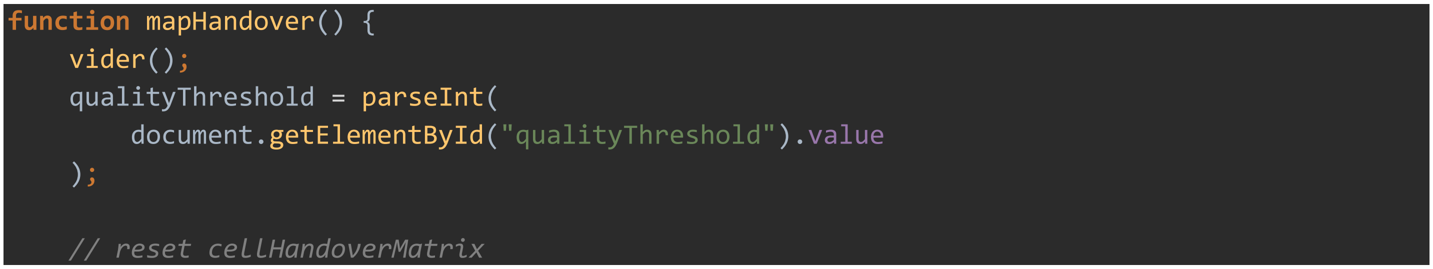
Links for the code:

GitHub repository - https://github.com/Hovhannes1/mobile-network-plan

Working link - https://hovhannes1.github.io/mobile-network-plan/

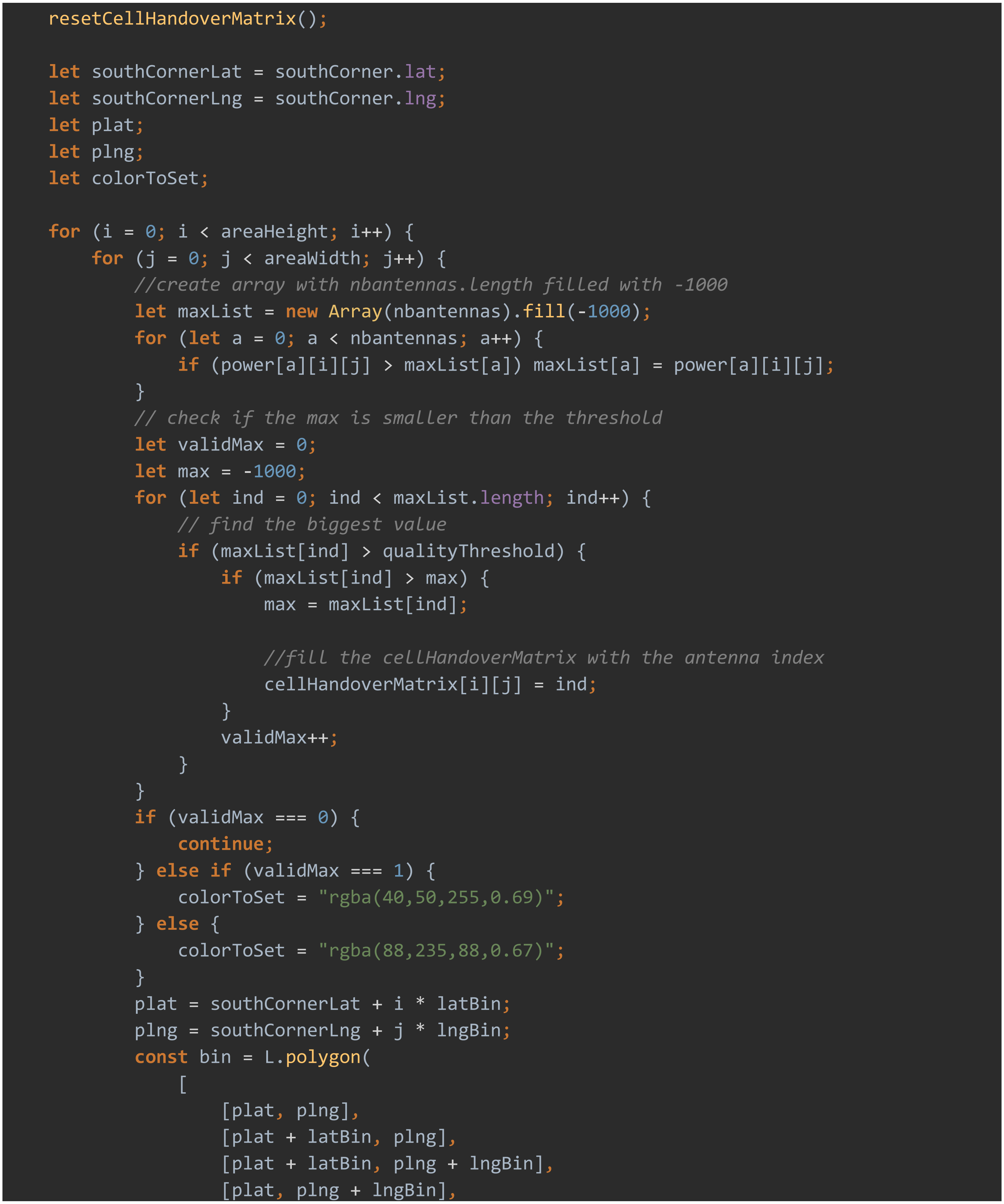
Main algorithms (code):

1. *Handover part*



```
function mapHandover() {
    vider();
    qualityThreshold = parseInt(
        document.getElementById("qualityThreshold").value
    );

    // reset cellHandoverMatrix
```



```
    resetCellHandoverMatrix();

    let southCornerLat = southCorner.lat;
    let southCornerLng = southCorner.lng;
    let plat;
    let plng;
    let colorToSet;

    for (i = 0; i < areaHeight; i++) {
        for (j = 0; j < areaWidth; j++) {
            //create array with nbantennas.length filled with -1000
            let maxList = new Array(nbantennas).fill(-1000);
            for (let a = 0; a < nbantennas; a++) {
                if (power[a][i][j] > maxList[a]) maxList[a] = power[a][i][j];
            }
            // check if the max is smaller than the threshold
            let validMax = 0;
            let max = -1000;
            for (let ind = 0; ind < maxList.length; ind++) {
                // find the biggest value
                if (maxList[ind] > qualityThreshold) {
                    if (maxList[ind] > max) {
                        max = maxList[ind];

                        //fill the cellHandoverMatrix with the antenna index
                        cellHandoverMatrix[i][j] = ind;
                    }
                    validMax++;
                }
            }
            if (validMax === 0) {
                continue;
            } else if (validMax === 1) {
                colorToSet = "rgba(40,50,255,0.69)";
            } else {
                colorToSet = "rgba(88,235,88,0.67)";
            }
            plat = southCornerLat + i * latBin;
            plng = southCornerLng + j * lngBin;
            const bin = L.polygon(
                [
                    [plat, plng],
                    [plat + latBin, plng],
                    [plat + latBin, plng + lngBin],
                    [plat, plng + lngBin],
                ],
                {fillColor: colorToSet, fillOpacity: 1, weight: 0, color: "none"}
            ).addTo(mymap);
            bins.push(bin);
        }
    }
}
```

2. *Cell movement and selection part*

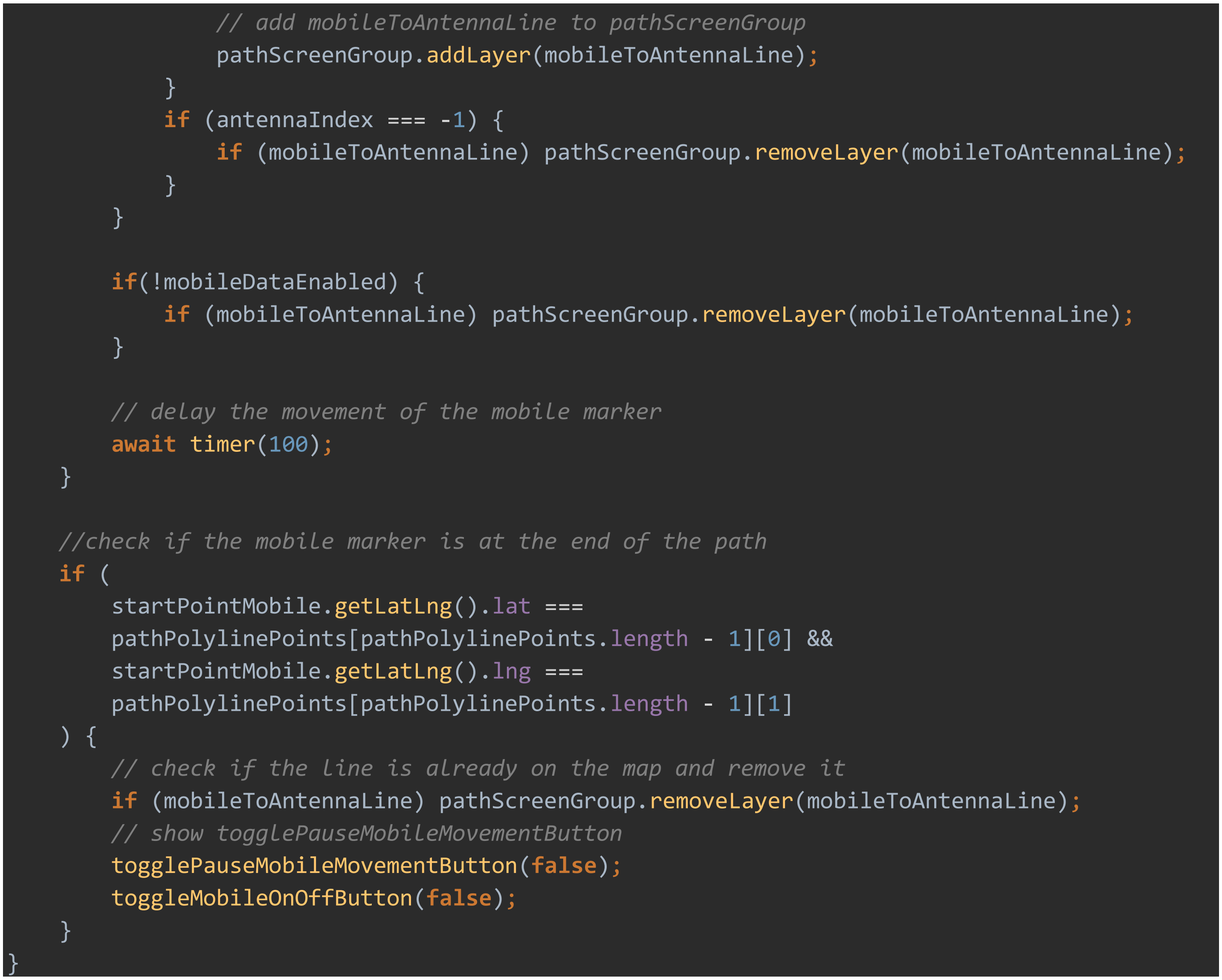
```
async function moveMobile() {
    // create a line connecting the mobile to antenna
    let mobileToAntennaLine;
    // calculate the avarage speed
    let time = 300;
    let avgSpeed = getPathLength() / time;
    // check if the mobile marker is not already at the end of the path
    while (
        startPointMobile.getLatLng().lat !==
        pathPolylinePoints[pathPolylinePoints.length - 1][0] &&
        startPointMobile.getLatLng().lng !==
        pathPolylinePoints[pathPolylinePoints.length - 1][1]
        ) {
        // check if marker not moving stop the loop
        if (isRestarting || isPaused) {
            if (mobileToAntennaLine) pathScreenGroup.removeLayer(mobileToAntennaLine);
            break;
        }
        // calculate dx and dy from startPointMobile to the next point
        let dx =
            pathPolylinePoints[currentCheckPointIndex + 1][0] -
startPointMobile.getLatLng().lat;
        let dy =
            pathPolylinePoints[currentCheckPointIndex + 1][1] -
startPointMobile.getLatLng().lng;

        // calculate the angle of the line
        let angle = Math.atan2(dy, dx);
        // calculate new lat and lng
        let newLat = startPointMobile.getLatLng().lat + avgSpeed * Math.cos(angle);
        let newLng = startPointMobile.getLatLng().lng + avgSpeed * Math.sin(angle);

        //check if the distance between current possition and the new possition is more than
        //the distance between the current possition and the next checkpoint
        let distanceBetweenInitialPosAndCheckpoint = Math.sqrt(
            Math.pow(
                startPointMobile.getLatLng().lat - pathPolylinePoints[currentCheckPointIndex
+ 1][0],
                2
            ) +
            Math.pow(
                startPointMobile.getLatLng().lng - pathPolylinePoints[currentCheckPointIndex
+ 1][1],
                2
            )
        );
        let distanceBetweenInitialPosAndNewPos = Math.sqrt(
            Math.pow(startPointMobile.getLatLng().lat - newLat, 2) +
            Math.pow(startPointMobile.getLatLng().lng - newLng, 2)
        );
        if (
            distanceBetweenInitialPosAndCheckpoint <
            distanceBetweenInitialPosAndNewPos
        ) {
            // move the marker to the next checkpoint
            startPointMobile.setLatLng([
                pathPolylinePoints[currentCheckPointIndex + 1][0],
                pathPolylinePoints[currentCheckPointIndex + 1][1],
            ]);
            //increase the currentCheckPointIndex
            currentCheckPointIndex++;
        } else {
            // move the mobile marker to new possition
            startPointMobile.setLatLng([newLat, newLng]);
        }

        // check if mobile data is turned on then connect to antenna
        if (mobileDataEnabled) {
            // check where is the marker on cellHandoverMatrix
            let antennaIndex = getCellIndex(startPointMobile.getLatLng());
            if (antennaIndex !== -1) {
                // check if the line is already on the map and remove it
                if (mobileToAntennaLine) pathScreenGroup.removeLayer(mobileToAntennaLine);

                // create a line connecting the mobile to antenna
                let antennaPos = [antennas[antennaIndex].location.lat,
antennas[antennaIndex].location.lng];
                mobileToAntennaLine = L.polyline([startPointMobile.getLatLng(), antennaPos],
{color: 'red'});
```



```
                // add mobileToAntennaLine to pathScreenGroup
                pathScreenGroup.addLayer(mobileToAntennaLine);
            }
            if (antennaIndex === -1) {
                if (mobileToAntennaLine) pathScreenGroup.removeLayer(mobileToAntennaLine);
            }
        }

        if(!mobileDataEnabled) {
            if (mobileToAntennaLine) pathScreenGroup.removeLayer(mobileToAntennaLine);
        }

        // delay the movement of the mobile marker
        await timer(100);
    }

    //check if the mobile marker is at the end of the path
    if (
        startPointMobile.getLatLng().lat ===
        pathPolylinePoints[pathPolylinePoints.length - 1][0] &&
        startPointMobile.getLatLng().lng ===
        pathPolylinePoints[pathPolylinePoints.length - 1][1]
    ) {
        // check if the line is already on the map and remove it
        if (mobileToAntennaLine) pathScreenGroup.removeLayer(mobileToAntennaLine);
        // show togglePauseMobileMovementButton
        togglePauseMobileMovementButton(false);
        toggleMobileOnOffButton(false);
    }
}
```