Below is represented the “bill's algorithm”. Before the driver ends the ride and exits the car, the system starts checking the state of sensors, the position of the car towards the position of the nearest safe area and last but not least the state of the battery ( the driver has 5 minutes to eventually charge the battery and receive the discount).The events generated and their consequences are discussed in the following table.

**[Legend]**

* + - *D*: driver;
    - *S*: System;
    - *C*: car;
    - *B*: battery;
    - LoP: list of passengers;
    - sA: safe area

|  |  |
| --- | --- |
| **Event** | **Consequences** |
| *D* exits C | *S* starts checking |
| Check the distance between the  sA and the current position | If((sA.nearest()-D.currPos())>3)  {D.applyTax()}  Else  {D.applyDiscount()} |
| *Check the number of passengers* | If(LoP.size()>2)  {D.applyDiscount()} |
| *Check the battery state* | If(B.getState()<20)  {D.applyTax()}  If(B.getState()>50)  {D.applyDiscount()} |
| *D* ends the rent | *C.* status = ”Ready” |
| D has 5 minutes to charge the car and take a discount | oldState=B.getState()  wait(5)  if(B.getState>oldState)  D.applyDiscount() |

Below is represented the “reservation/rent algorithm”. The algorithm starts when a user clicks on the map provided by the system; immediately the controller of the system hides the selected car and starts the time of an hour (maximum amount of time that the user can wait before starting the rent). If the time exceeds the fixed constraint, the car returns available on the map, otherwise the status is "rented" (because it means that the user has pressed the start button and the ride can begin).

**[Legend]**

* + - *U*: user;
    - *R*: reservation;
    - *C*: car;
    - *S*: system;

|  |  |
| --- | --- |
| **Event** | **Consequences** |
| *U* selects car on the map | *C.status = “Reserved”*  *S.hideCar();* |
| Check the reservation’s time and compare it with the current time | If(system.getCurrTime()-R.time()>1)  {C.status=”Free”;  U.applyTax()  S.showCar()}}  Else  {C.status=”Rented”} |
| *Compare the positions* | If((U.position-C.position)<1)  {myApp.enableStartButton()} |
| *User starts the engine* | C.startCharge() |

Below is represented the “geolocation’s algorithm”. Let’s begin with the premise that we are imagine to build the algorithm with an object oriented Language and we’re providing a pseudo-code. This algorithm checks if the given point is inside this Triangle. Infact thanks to a theorem about convex polygons, we can check if a point P is inside a given convex polygon (i.e. if the given vector associated to the point P is a convex combination of the polygon vertices). We can calculate if such coefficients exists solving a vector equation:

1. P = dX\* P1 + dY\* P2 + dZ\* P3; ( with P, P1, P2, P3 in R^2; with dX, dY, dZ in R;

dX1 >= 0, dY>= 0, dZ>= 0 and dX+dY+dZ = 1.)

2) P = dX \* P1 + dY\* P2 + (1 - dX- dY) \* P3.

3) P - P3 = dX\* (P1 - P3) + dY \* (P2 - P3).

This equation can then be split into two scalar linear equations in the x and y components. The system is solved using Cramer's rule and then it is checked that alpha1 and alpha2 (and alpha3) found by solving the system satisfy the constraints.

**[Legend]**

* + - *P*: class Point;
    - *T*: class Triangle;
    - *Z*: class Zone;
    - *A*: class Area;
    - *a,b,c,d,e,f*: Double(or Float) values;

|  |  |
| --- | --- |
| **Event** | **Consequences** |
| *Declare variables that will allow to solve the linear equation system thanks to Cramer’s method* | *a= p1.getX() – p3.getX()*  *b= p2.getX() – p3.getX()*  *c= p1.getY() – p3.getY()*  *d= p2.getY() – p3.getY()*  *e= p.getX() – p3.getX()*  *f= p.getX() – p3.getY()* |
| Calculate the determinant to check the solution of the system | tContains(){  d=a\*d-b\*c  if(d==0) return false  dPx = e\*d – f\*b  dPy = a\*f – c\*e  dZ = 1 – dPx – dPy  dX= dPx/d  dY= dPy/d} |
| *Check the results and draw conclusion* | if(dX<0 || dY<0 || dZ <0)  {return false}  return true; |
| *Instance the Zone Class and set the Zone as a set of Triangles. Then check if a point (in our case our position) is contained in the triangle* | zContains(){  for (Triangle t : triangles)  if (t.tContains(p))  {return true;}  return false;} |
| *Instance the Area Class and set the Area as a set of Zone. Then check if a point is contained in the zone. This allows us to determine in which zone is our point* | *Point(latitude,longitude)*  *for (Zone z : zones)*  *if (z.zContains(p))*  *return z;*  *return null;* |

\*the method tContains() should be defined in the Triangle Class as with for zContains() method that will be inside the Zone class