A moving target

When we try to solve this problem, we can use information to remove impossible cells where target must not be in and get a list of cells. Target must be in the cell of this list. Every time we get new information, we will use this information to review cells in the list, remove impossible cells and update list by replace cells with their neighboring cells. Then we choose one cell to search based on Rule 1 or Rule 2.Here are detailed steps.

1. **Initialize the list of cells**

At first, assume that target is in a given cell called *Ct1*. We have no information and all cells are with uniform probability. So we randomly choose a cell to search and get an information. Assume that now target moves to a neighboring cell called *Ct2*, and we get an information from this move, call it *Info1*. *Info1* has two contents, type of *Ct1* and type of *Ct2*. These two contents have no order, but for clearly representing, we represent *Info1* as *Info1* [type of *Ct1*, type of *Ct2*]. From *Info1*, we can remove lots of impossible cells. But at this point, we are still not sure what type of cells where target is in. So, for efficiency, we think that we should randomly choose another cell and get second information.

After our second search, if we didn’t find target, target will move from *Ct2* to a neighboring cell *Ct3*. And we get second information *Info2* [type of *Ct2*, type of *Ct3*]. By comparing *Info1* and *Info2*, we have big chance to confirm what type of *Ct2* is. We search *Info1* and *Info2* to find what type appears twice.



There is only exception like [flat, hilly] and [flat, hilly], in this exception we will get two types. In other circumstances, we will get only one type. Then we traverse all cells in the map, to find cells *P* whose type is the type appears twice in *Info1* and *Info2*, also these cells have at least one neighboring cell whose type is the other type in *Info1* and at least one neighboring cell whose type is the other type in *Info2*. Then we will use a list to record all cells satisfied. *Ct2* must be in this list, then we traverse all cells in the list, find and record all its neighboring cells whose type is the other type in *Info2*. Then we get another list *Points* [P1, P2, …,Pn ]. Now *Ct3* must be in the list *Points.* Then we give a value to all cells in the list according Rule 1 or Rule 2. When using Rule1, we get a list like *PointsRule1* [(P1, 1), (P2, 1),…, (Pn,1)]. When using Rule 2, we get a list like *PointsRule2* [(P1,1-fn), (P2, 1-fn),…, (Pn,1-fn)], fn is the false negative of this cell. And choose one cell with highest value to apply our third search. If we fail to catch target, the value of this cell multiplies its false negative and get third information.

1. **Update list to neighboring cells**

From third search, we will use same update strategy. When applying Rule 1 or Rule 2, we use the same update strategy. Assume that now we get *Infoi*[type of *Cti*, type of *Cti+1*] and a list *Points* of cells with values that *Cti* must be in it. Firstly, we backup *Points* to a new list called *PointsBackup* and empty list *Points.* Then traverse all points in *PointsBackup,* compare type of this point *C* and types in *Infoi*. If type of this point *C* isn’t same as any one of *Infoi,* move to next point. If type of this point *C* is same as one of *Infoi*, then we compare types of all its neighboring cells and the other type in *Infoi*. If type of a neighboring cells is same as the other type in *Infoi*, we record it in the list *Points* with a value that is the value of point C divides number of satisfied neighboring cells of point C.



After testing all points in *PointsBackup, we will get an updated list Points.* But there is a problem with this list *Points*, so we need another step.

1. **Combination same cells**

By our experiment, we find that some points will appear several points after step 2. Assume that we get an information [flat, hilly] and there is a point W with type ‘flat’. Types of its neighboring cells are ‘hilly’, ’hilly’, ’hilly’ and ‘flat’, all its neighboring cells with type ‘hilly’ are in list *Points.*



When we test its first neighboring cell and find this point satisfied, we record it. Then we test its next neighboring cell and record this point. Again, after testing, we record this point. Then for this one point, it appears third times in list *Points*. This will waste lots of time for next turn of finding satisfied neighboring cells. So we need to combine same cells in the list *Points* to shorten length of list *Points.* In the circumstance, we talked about above. Assume that probability of target in its first neighboring cell is p1 and this neighboring cell has x satisfied neighboring cells, probability of target in its second neighboring cell is p2 and this neighboring cell has y satisfied neighboring cells, probability of target in its third neighboring cell is p3 and this neighboring cell has z satisfied neighboring. Then probability of target in point W is (p1/x+p2/y+p3/z). Before combination, point W appears third times in list *Points* with values p1/x, p2/y and p3/z. After combination, point W appear only once with value (p1/x+p2/y+p3/z). With combination, we make sure that list *Points* won’t be so long.

After step 3, we choose one cell with highest value to search. If we fail to catch target, the value of this cell multiplies its false negative and get next information. Then repeat step 2 and step 3 to find which cell to search and search the cell until we catch the target.

In our experiment, every turn we first apply Rule 1 and target moves randomly, we record target’s moving path until target is caught. We apply Rule 2, target will move in the same path as recorded moving path until there is no points in moving cell, then target will move randomly until target is caught. We experiment 1000 turns and find that when applying Rule 1, average steps to catch the target is 8891, when applying Rule 2, average steps to catch the target is 8760. Generally speaking, performances of two rules are same.

But one of group members has a different opinion. He thinks that policeman is applying Rule 1 that policeman compare P (keys in the park) and P (keys in light). However, the drunk compares P (keys found in light| keys in light) and P (keys found in the park| keys in the park). The drunk has replied that he lost keys in the park, so he know that there is a higher probability that key is in the park, but he still search in light means that he doesn’t take P (keys in the park) and P (keys in light) into consideration. What the drunk replies at last is that the light is better here. This sentence only talk about visibility and this is why he search here, which means that the drunk choose where to search, depends on the visibility. And visibility expressed in mathematical expression is P (keys found in here| keys in here). So, the drunk compares P (keys found in light| keys in light) and P (keys found in the park| keys in the park) and decide to search in light.