Bombard the Warship

Wentao Fan
CS 5114 Theory of Algorithm
Project 1 Report, 03/02/2020
wentaofan@vt.edu

1. Aim

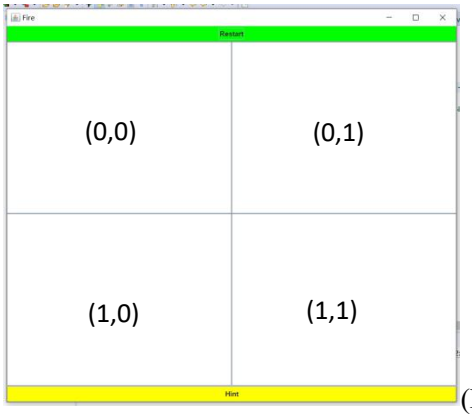
Recursive method is always used when meeting problems which needs a lot of calculation to get the result. Recursive method can save space of writing codes but it wastes running time as it tends to repeat processes of running. In contrast, Dynamic Programming takes another way in solving problems which costs more space but saves more time. Dynamic Programming can be very efficient when it meets a large amount of calculation.

This project is going to implement an optimal way (Dynamic Programming) to find out the target in a 2-D map. You can see it as a game of "Bombard the Warship". Player goal is to bombard the ship within the 2-D map. The fewer steps the player used, the higher score the player will achieve. During the game, player can ask for hint at the start of the game and the hint will show the optimal choice to bombard the ship.

2. Method

Suppose that there is some amount of targets existing inside a 2-D map in any size of m*n(m>=2 && n>=2). A ship is represented as length of s and width of 1(2<=s<=5). The ship can deploy in the map in either horizontal or vertical direction. Each time of shooting, the player can choose a coordinate of (x,y) (0<=x<=m && 0<=y<=n) as a shoot. When a ship is shot once, it is destroyed and the player gets corresponding score. Each time of shoot costs a bullet. For different size of map, bullets available is different. The game will end in 3 conditions: 1. All the ships are hit once, the player wins; 2. When clicking more than half of the number of all cells without hitting any ship, the player loses; 3. When clicking more than half of the number of all cells, the player does hit some of the ship(not all), it is "winning in narrow margin".

The easiest 2*2 map



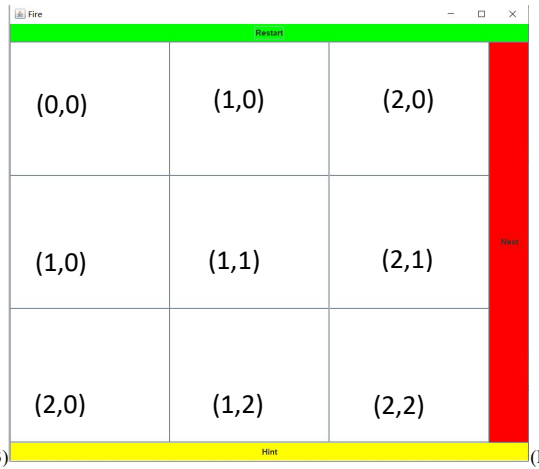
(Fig 1)


(Fig 2)

Let the most up-left cell be the origin(0,0), i be the vertical variable and j be the horizontal variable. The smallest map is in the size of 2*2, and the smallest ship size is 2. In this smallest map, the ship has 4 choices of its position in total: {(0,0),(0,1)}, {(0,0),(1,0)}, {(0,1),(1,1)} and {(1,0),(1,1)}. Then there is no difference of the first clicking as each cell enjoys a hitting possibility of 50%. Numbers in

Figure 2 show that how many conditions are possible for the ship in each cell. For example, (0,0) may have ship of {(0,0),(0,1)} or {(0,0),(1,0)}, which is 2 conditions in total. The rest cells are in the same reason.

(Fig 3)



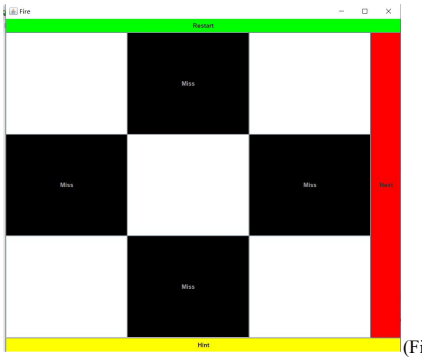
(Fig 4)

After the first click on(0,0) there is going to have 2 cases: hit or miss, each is 50% possibility. If it is missed, then you will find that the numbers of condition in each cell change now. (0,1) deducted from 2 to 1, this is because (0,0) has been attacked and we find no ship here, so {(0,0),(0,1)} is no more possible, only{(0,1),(1,1)} left for (0,1). As the same reason, (1,0) is also 1.

Now only (1,1) still has 2 conditions because its ship possibility is not affected when (0,0) is checked. Then for the second click, hit in (0,1) or (1,0) is both 50%, and hit in (1,1) is 100%! So that the second click must hit the ship. Then for map 2*2, ship length 2, best case is 1, worst case is 2. As each cell in this case share the same possibility, Recursive method has no difference with Dynamic Programming here.

3*3 map

The 3*3 map is a little more complex and from this case, Recursive and DP can show some difference in solving the problem. We still talk about the case of ship length 2 here.



(Fig 5)

(Fig 6)

In each row, the ship has 2 possible choice to place. e.g. for the first row, the ship can place in {(0,0),(1,0)} or {(1,0),(2,0)}. In order to fire on the ship, each shoot can leave space of at most 1 cell in a row(If leaving 2 cells it may miss a ship). How many cells to leave depends on the ship length. It should always leave a space of length-1 cells to avoid miss a ship. So that the best way for the first row is to fire on (1,0), which costs only 1 shoot when ensuring row 1 has no ship.

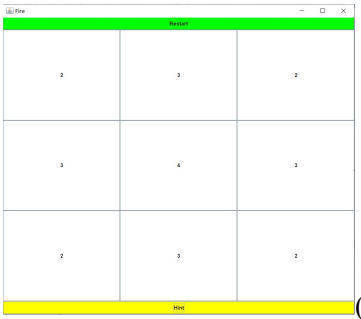
Recursive

The condition is the same for the columns: leaving no more than 2-1=1 cell space. Row1 fires from the second cell, so that row 2 has to fire from the first cell. Then row 3 fire from the second and so

on(). Each time of fire in the row is from the column of length-1, length-2, length-3...length-1, length-2, length-3...When length-n reaches 0, then next time reset the value to length-1. The Recursive result is like Figure 5, where the black cells from up to down shows the recursive solving way. Figure 6 shows that when a ship is hit, the whole ship expose in blue with “Hit!!”.

Dynamic Programming

Recursive method is an easy-to-understand way but not the most efficient way, because it treats all cells equally to find the target. However, in fact, the probability of ship placing is no more equal in different cells. In each row, ship of length 2 can have 2 ways of placing. This is also the same for each column. As this map is 3*3, then it have 3*2+3*2=12 ways of placing. Then we can draw a diagram like Figure 2 which shows how many possible ways that each cell has, which is like Figure 7.


(Fig 7)

(Fig 8)

Figure 7 shows that how many possible placings are in each cell. A ship cost 2 cells, so the sum of all the cells is equal to 12*2=24. It is easy to find that the middle cell has the most possible placing 4, which means that the probability of hitting the ship in this cell is 4/12=33.3%! If firing on a corner cell instead, probability of hitting will be only 2/12=16.7% which is just half of the middle cell! So that it is optimal to shoot the middle cell for the first step. Then the map shows value change in Figure 8. After the first shoot, all the rest cells are 2 now, which means that their properties are now equal. Then shoot 4 cells in cross or diagonal so that this case is then solved. This requires 5 shoots in total for the worst case.

Large Problem

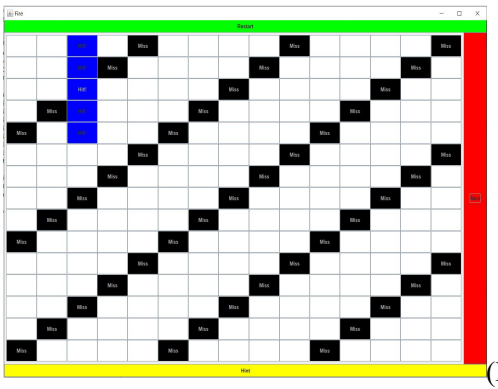
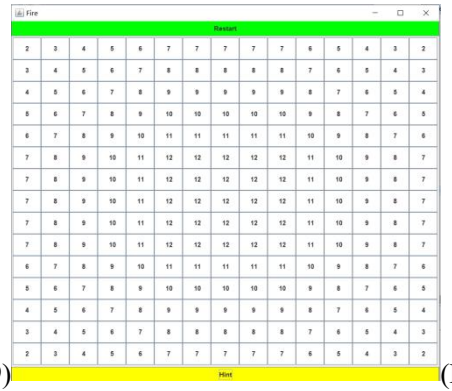
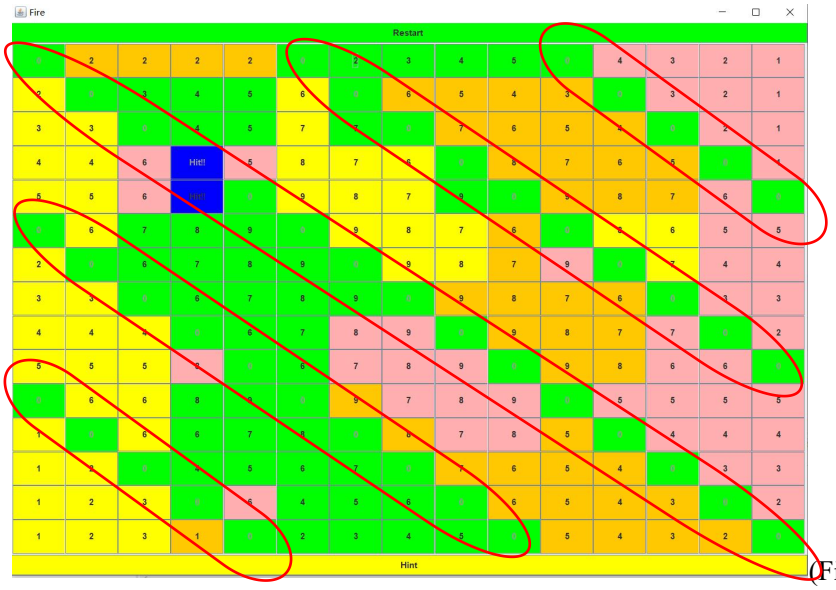

(Fig 9)

(Fig 10)

Figure 9 shows a recursive method to solve a 15*15 size map and ship length=5. Each row have 3 shoots and the total worst run time is 15*3=45.

Figure 10 and 11 shows how DP Top-Down solves this problem. The red circles shows all the cells(the green 0s) which it has to fire on for the worst case. Although the worst case is still 45, but it has a much higher rate to find the solution faster than recursive method, because DP chooses the highest number at every time.



(Fig 11)

The difference of Top-Down and Bottom-Up is that Top-Down starts firing on the largest cells, and the Bottom-Up starts firing on the smallest cells. All the cells that they have to fire is the same until the ship is hit. Bottom-Up starts firing from the corner cells, then to the cells on sides, and then on cells in middle gradually.

3. Algorithm

Let S be the solution. Let x and y be row and column of the map.

	Top-Down(every # of S can be more than 1)	**Bottom-Up**	**Recursive**
Test Case:(15 *15, length=5)	25,17,32,21,10,8,16 Average:18.43	18,35,32,23,41,29,36 Average: 30.57	10,42,27,36,18,35,21 Average: 27.00
Algorithm	S1=[x/2,y/2] S2=[x/2 ± 1,y/2 ± 1] S3=[x/2 ± 2,u/2 ± 2] ... Sn=[x/2 ± n,y/2 ± n]	S1=[0,0],[0,y],[x,0],[x,y] S2=[1,0],[0,1],[1,y],[0,y-1],[x,1],[x-1,0],[x-1,y-1] S3=[2,0],[1,1],[0,2],[2,y],[1,y-1]... ... Sn=[x_{Sn-1} ± 1, y_{Sn-1} ± 1]	j=y-1, i=x-1 S11=[i,j+shipLength-1], s12=[i,j+2shipLength-1], s13=[i,j+3shipLength-1]... s1n=[i,j+nshipLength-1] S21=[i+1,j+shipLength-1], s22=[i+1,j+2shipLength-1],... s2n=[i+1,j+nshipLength-1]; ... Snn=[i+n,j+nshipLength-1]
Pseudo-code	Top-down(i,j,shipLength) Let S[1...n] be the list i=0 to x/2	Bottom-up(i,j,shipLength) Let S[1...n] be the list i=0 to x/2 j=0 to y/2	Recursive(i,j,shipLength) Let S[1...n] be the list i=0 to x j=0 to y

	j=0 to y/2
	S.add([i+x/2, j+y/2])
	return (i ± 1, j ± 1, shipLength)
		S.add([i,j])
		S.add([x-i,y-j])
		S.add([x-i,j])
		S.add([x,y-j])
		return ([i ± 1, j ± 1], shipLength)
			j+=shipLength-1
			xl=j
			for a=0 to x-1
			for b=xl to y-1,b+=shipLength
			S.add([a,b])
			endfor
			xl--;
			if xl<0
			xl= shipLength-1
			endif
			endfor
			return S

O(n)	Runtime<=n/shipLength. As shipLength is a constant, total is O(n)	Every time there is an increase of 1 time for calculation. Run time>=n^2/shipLength, which is O(n^2)	Run time =n^2/shipLength. As shipLength is a constant, total is O(n^2)
Ω (n)	Ω (n)	Ω (n^2)	Ω (n^2)
Θ (n)	Θ (n/2)	Θ (n^2)	Θ (n^2)

The fastest method is DP Top-Down. As every running time it selects the most possible choices, it has the highest possibility to find the solution earlier.

The second is Recursive. It treats cells equally so that it is slower to find the solution. Because it does not know how much possibility that the choices it selects every time.

DP Bottom-Up is the last one. Searching from corner and side is the slowest way as it contains most of the lowest possible choices. Recursive does not always meet the low choices, but Bottom-Up does.

4. Conclusion

As the value of length, width of the map and the size of the ship are able to change to any rational value, we could find something interesting. When the size of map is limited to not very large value, the ship has a high rate to appear around the middle of the map other than corners and sides. However, when increasing the map size and making it much larger than the size of the ship, its possibility of appearance become equally distributed inside a very large range of “equally distributed area” in middle of the map(e.g.in Fig 10 you can find a large square of “12”s). This means that when the area is increased to unlimited amount, the “equally distributed area” will also become much larger(this is like looking for a ship in Pacific Ocean). Suppose that if there is no other interference, DP-Top-Down and Recursive methods have almost the same efficiency when searching target in the “equally distributed area”(When changing all the cell values to a same value, there will be little difference between the 2 methods).

DP-Top-Down is efficient when the elements of the map is well known. As it runs Top-Down based on specific cell values which shows difference of cells, without such value DP will have nothing to do then. Recursive is suitable when the map is not well known. When each cell means no difference, recursive is then very effective. The concrete details decide which method to use in reality.