a) We have completed the project. Our code runs successfully on all example inputs.
b) We choose to split the processors checkered. Checkered implementation runs faster and it was challenging, we enjoyed it while coding.

- We create an NxN 2D array that contains all of the board, and send parts of maps to each processor.
- Coordinates of towers are read from the input file and pushed to the array. The algorithm finds each tower's processor and stores it to the array and sends it to the corresponding processor at the start of each wave. Now everything needed is sent from the manager to the processors.
- Each processor first has to take its row, column, wave round and part of the map that it is responsible for. After that, the for loop iterates the number of wave times to simulate each wave. It receives placement of towers at wave starts. If the coordinate is empty it replaces it with the new tower.
- Each processor executes 8 rounds per wave. Each tower should know how much damage it will take from neighbor towers. To do this, a processor should know its surrounding processors' maps to control towers around. To do this,
  - even rows send bottom side to below odd and wait for a response from it.
  - odd rows send bottom side to below even and wait for a response from it.
  - odd columns send their right sides to right and wait for a response from it
  - even columns send their right sides to right and wait for a response from it

But processors do not have cross neighbors. We get the cross neighbors' cells from its right and left neighbors' top and bottom neighbors since they have that information from former sends and receives.
After finding all neighbors of processors, we create an array that contains cells and side neighbors to check how much damage a tower gets, and decrease the health of the tower. This procedure executes for each round.

At the end of the waves, we concatenate the part maps to create the whole map.

c) Complexity is approximately equal to the communication complexity. So, it is convenient to count send operations. At manager processor

```
for i in range(1, P+1):
        row = (i-1) // sqrt_p
        column = (i-1) % sqrt_p
        part_map = [[column for column in row[column * sqrt_cell:(
            column+1)*sqrt_cell]] for row in game_map[row * sqrt_cell:(row+1)*sqrt_cell]]
        comm.send({"row": row, "column": column, "wave": wave,
                   "part_map": part_map}, dest=i, tag=0)
```

```
for i in range(wave):
        wave_count = i
        process_towers = [[] for x in range(size)]
        o_tower = o_s[i]
        plus_towers = plus_s[i]
        for o in o_tower:
            process_towers[find_myself(
                o["col"], o["row"], sqrt_cell)-1].append(o)
```

```
    for plus in plus_towers:
        process_towers[find_myself(
            plus["col"], plus["row"], sqrt_cell)-1].append(plus)
    for ad1 in range(1, P+1):
        comm.send(process_towers[ad1-1], dest=ad1, tag=22)
```

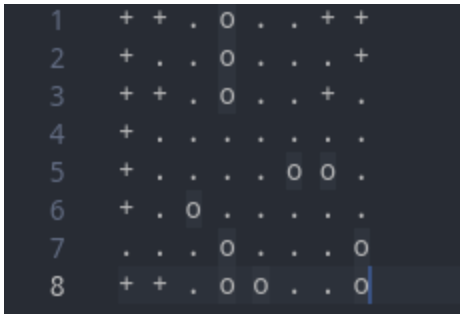
this two send operations are count and found
(P)+(P)*wave
For each worker processor at each wave 8 round is executed and 4 send operations perform so, wave*8*4. In addition there is one more send operation at the final step, operation to send final maps to the manager

F(P, wave)=(P)+(P)*wave+wave*8*4+1 ~ O(P*wave)

d) Outputs:

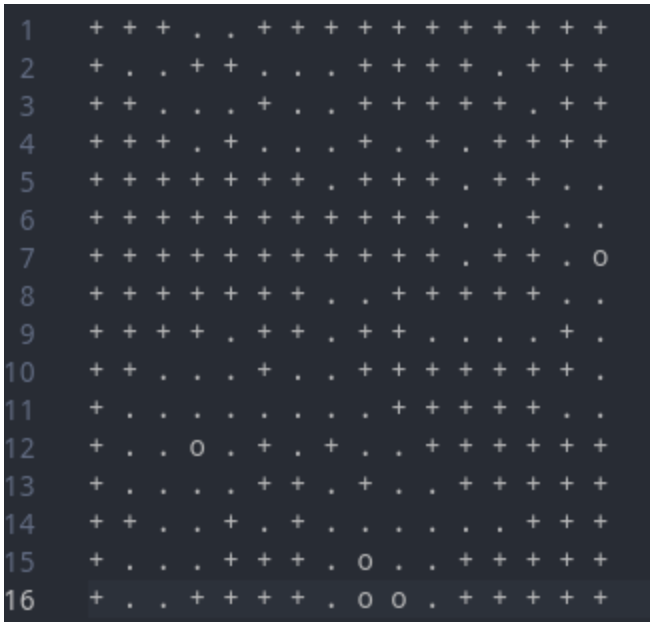
- input1:

- input2:

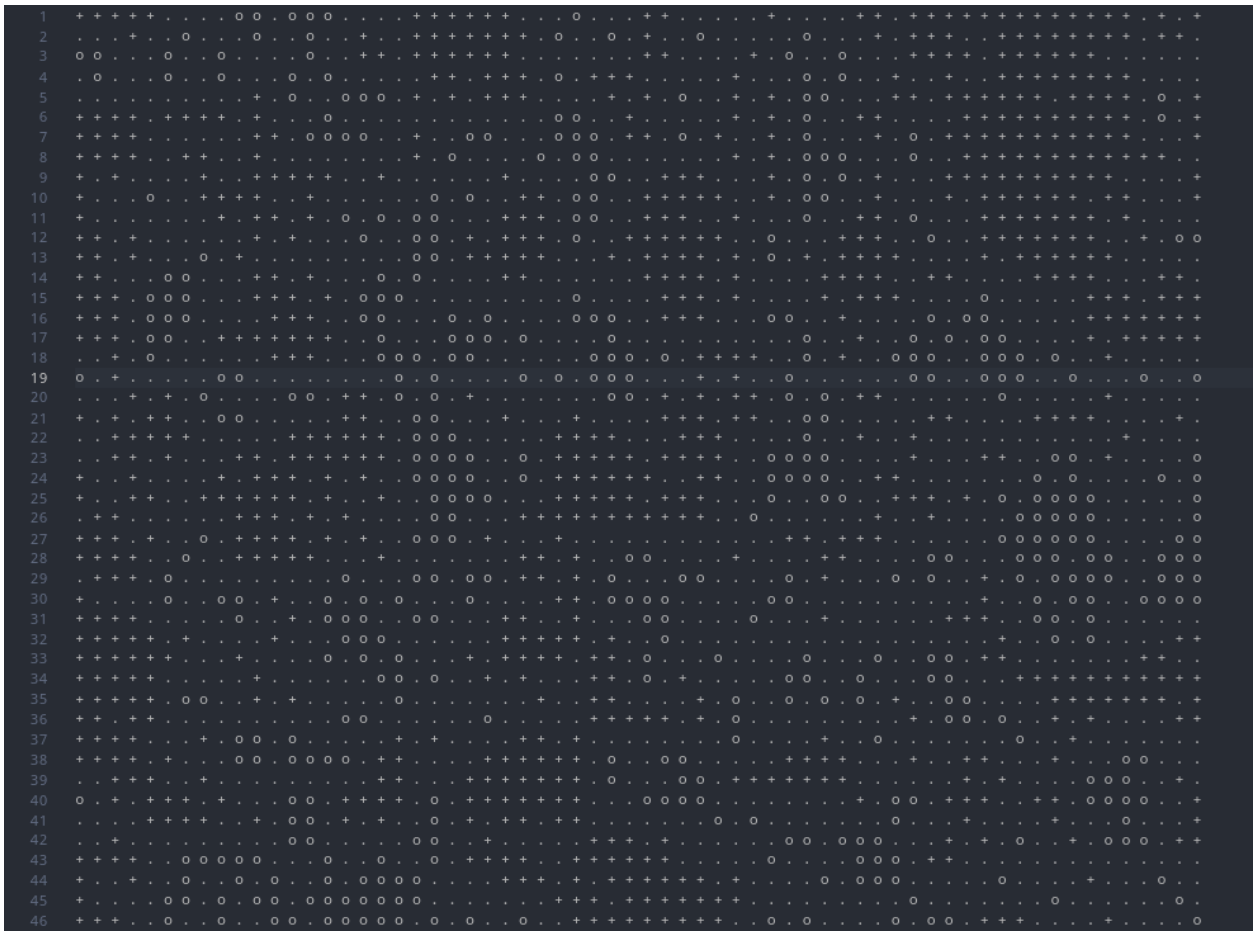


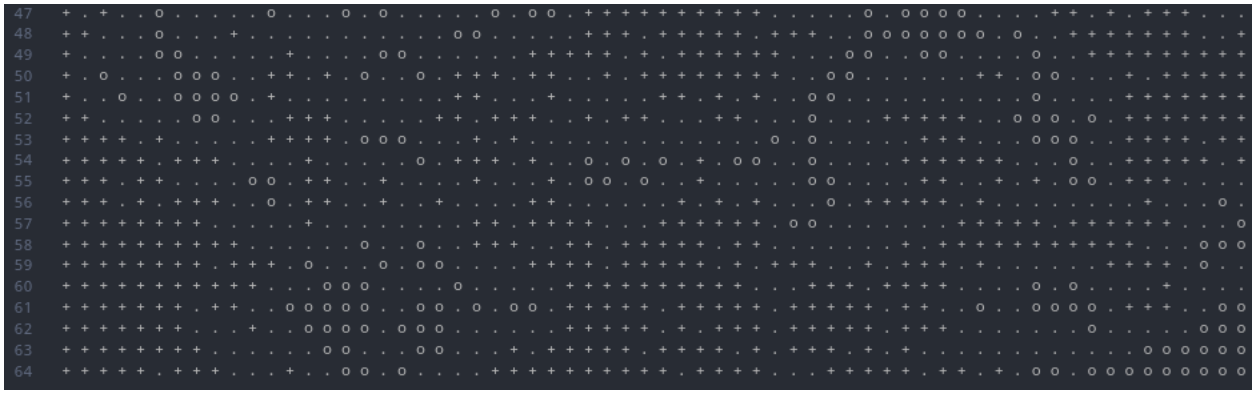
- input3:

- input4:



- input5:





e) Difficulties and Conclusion:

As you can guess, the attack part was pretty straightforward after getting all the neighbors. However, until that point designing the communication part was not easy. We had many ideas and evaluated them in terms of how easy it is to implement, how it will perform and code reusability. Our approach for communicating cross neighbors was harder to implement and debug; however, it performs faster than having 4 more send-receive operations for each cross neighbor, 8 in total. We could speed things up a little more by using mpi.Send function instead of mpi.send function but our implementation runs pretty fast for the given test cases