Project 02
Treasure Island

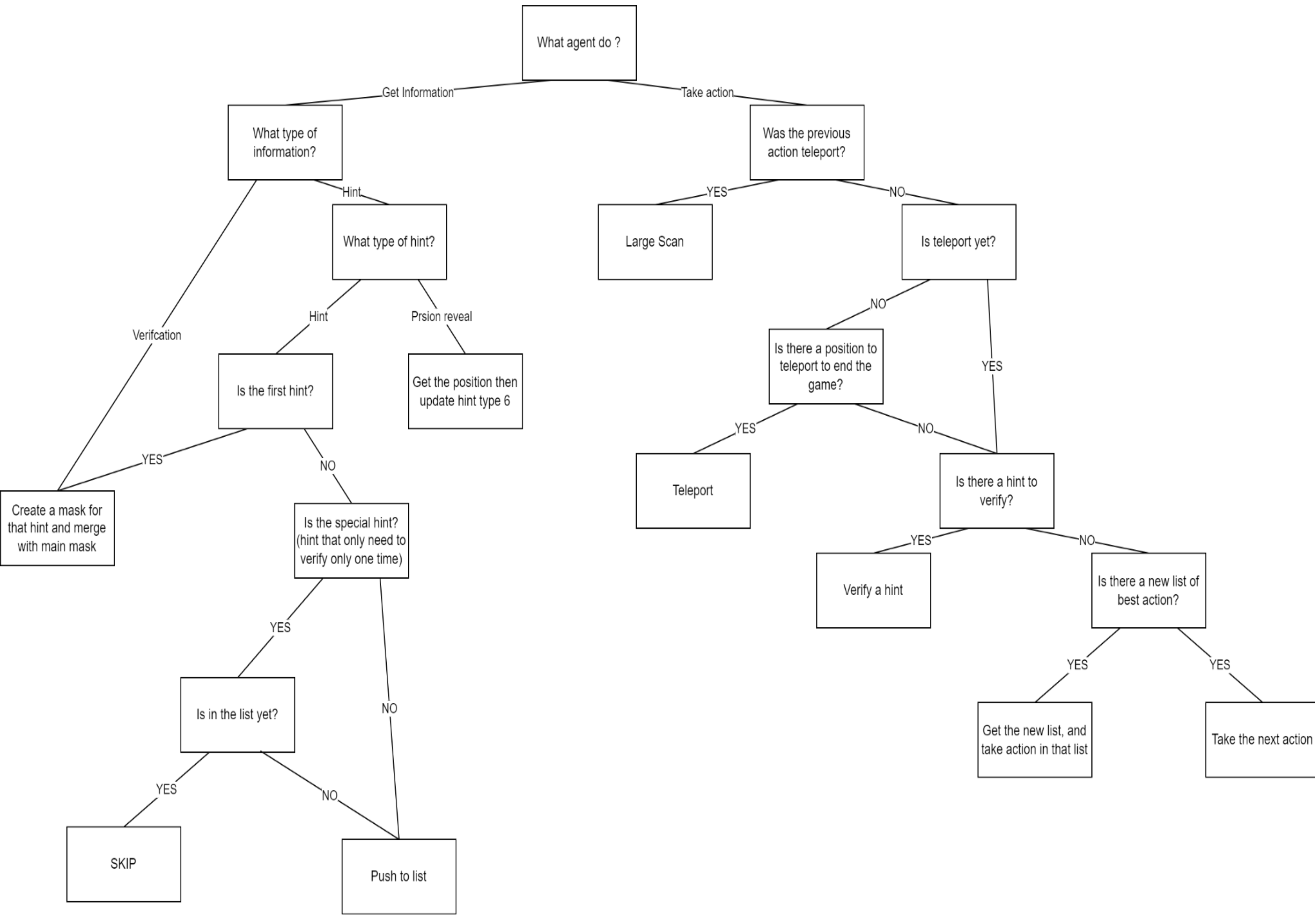
Member:
20125010 Trần Bảo Lợi
20125023 Vũ Hoàng Đạt
20125032 Trương Việt Hoàng
20125069 Trần Lạc Việt

I. Self-assessment

No.	Criteria	Scores
1	Create 5 maps: - 3 maps with sizes: 16x16, 32x32, 64x64 - 2 maps larger than 64x64	10%
1.1	Implement a map generator	10% (bonus)
2	Implement the game rule	20%
3	Implement the logical agent	20%
4	Implement the visualization tools	15%
5	The agent can handle a complex map with large size, many of regions, prison, mountains	5%
6	Report your implementation with some reflection or comments	20%
7	Contest* (unknown)	
Total		100%

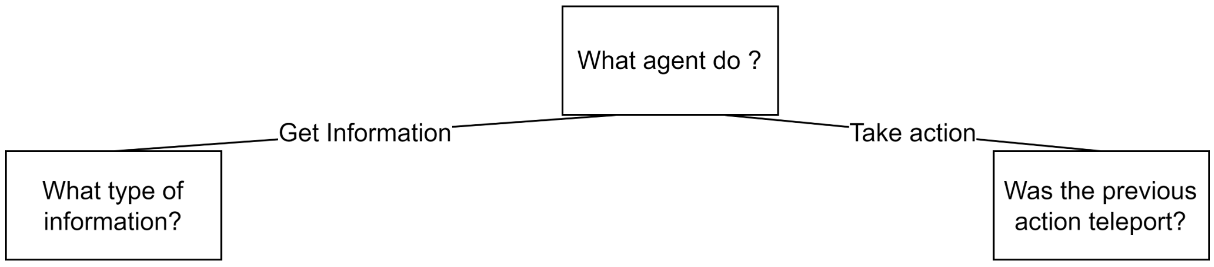
II. Algorithms' description

Overview:

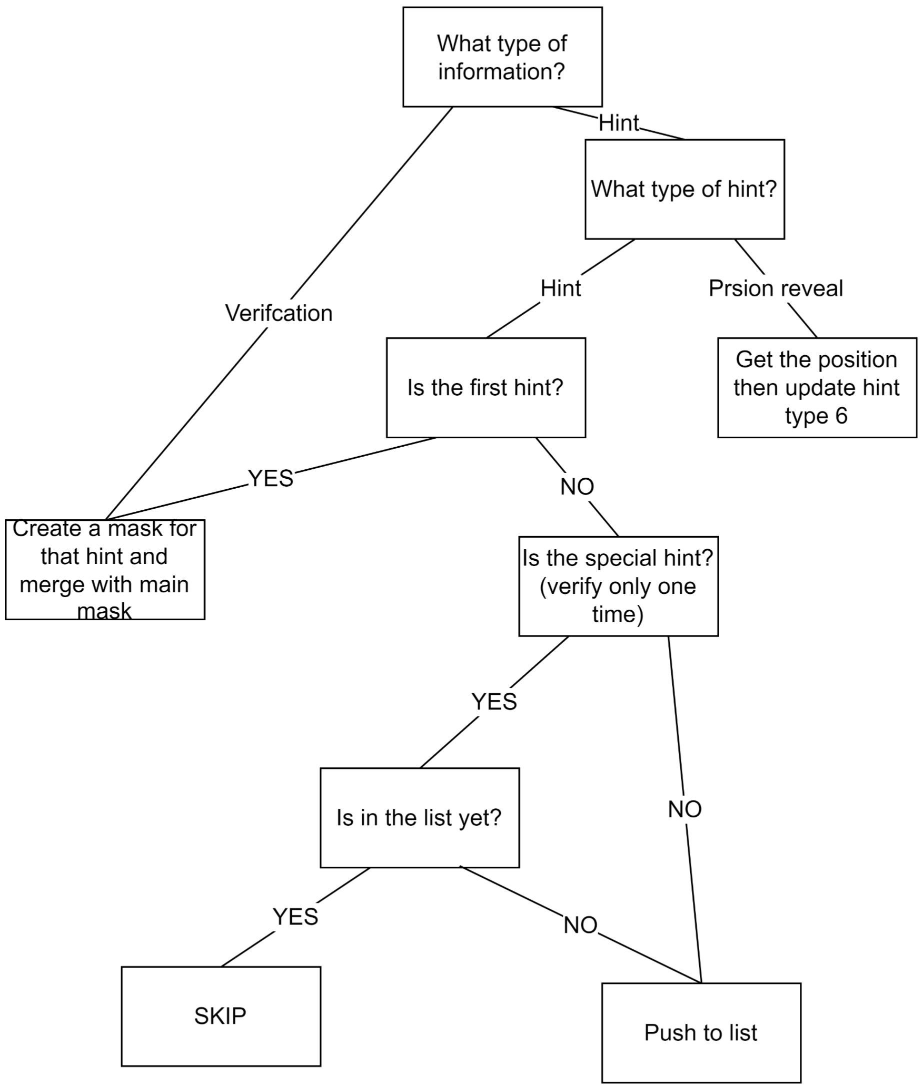

What agent do ?
Get Information
Take action
What type of information?
Hint
Verifcation
What type of hint?
Hint
Prsion reveal
Is the first hint?
YES
NO
Create a mask for that hint and merge with main mask
Get the position then update hint type 6
Is the special hint? (hint that only need to verify only one time)
YES
NO
Is in the list yet?
YES
NO
SKIP
Push to list
Was the previous action teleport?
YES
NO
Large Scan
Is teleport yet?
NO
YES
Is there a position to teleport to end the game?
YES
NO
Teleport
Is there a hint to verify?
YES
NO
Verify a hint
Is there a new list of best action?
YES
YES
Get the new list, and take action in that list
Take the next action

Details:

The agent has two main types of activities: Getting information and Taking action.



- **Getting information:**


What type of information?
Hint
Verifcation
What type of hint?
Hint
Prsion reveal
Is the first hint?
Get the position then update hint type 6
YES
NO
Create a mask for that hint and merge with main mask
Is the special hint? (verify only one time)
YES
NO
Is in the list yet?
YES
NO
SKIP
Push to list

Agent's main mask is used to save unknown places and certainly nothing there. We mark "1" for places that have scanned or are certain that there is no treasure there (by hint, mountain, sea) and "0" for places where we know nothing about.

The special case of hints:

The True/False hints:

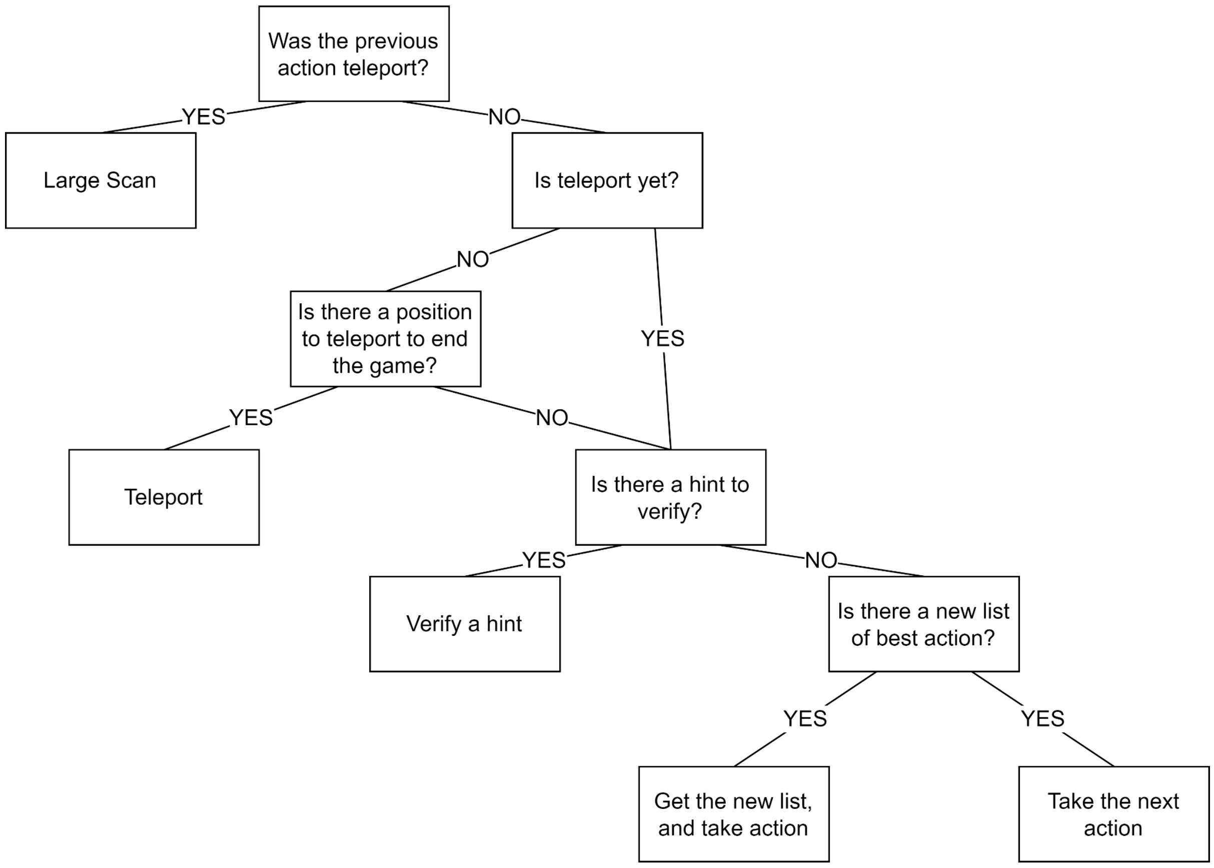
- Hint 15: The treasure is in a region that has Mountain.
- Hint 11: The treasure is somewhere in an area bounded by 2-3 tiles from Sea.
- Hint 10: The treasure is somewhere in a boundary of 2 regions.

These hints only need to verify 1 of each type, because those of the same type only have the same meaning.

Hint that need other information to create a mask:
Hint 6: He tells you that you are the nearest person to the treasure (between you and the prison he is staying in).
This hint can create a mask when we have a position of the prison, so for this type, we wait until the prison reveals then create a mask.

With confirmed hints including type 6 hints, we will mask that hint and merge it with the main mask. Summary of how to create a mask: mark the cells with closer Manhattan distance. Details on how to create a mask will be discussed in the following section.

- **Taking action:**


Was the previous action teleport?
YES
NO
Large Scan
Is teleport yet?
NO
YES
Is there a position to teleport to end the game?
YES
NO
Teleport
Is there a hint to verify?
YES
NO
Verify a hint
Is there a new list of best action?
YES
YES
Get the new list, and take action
Take the next action

There are 5 types of actions:

1. Verification, verify a hint is a truth or a liar.
2. Move straight 1-2 steps in a direction then perform a small scan.
3. Move straight 3-4 steps in a direction.
4. Stay and perform a large scan.

And the special action ONCE per game: the agent can teleport (instantly move to anywhere on the map EXCEPT tiles with label “0”, and “M”) and it is free to use (doesn’t cost 1 action in a turn).

The only way to 100% win this game is that scan every tile on the map before the pirate reaches the treasure

We see that teleporting does not consume action and when combined with a large scan will be a powerful tool to bring victory to the agent. But teleporting to the wrong location further than the treasure or into an area with no way to the treasure, the agent will definitely lose.

We suggest a strategy that will perform a teleport when it is certain that after that teleport, we can do a large scan with which all the locations on the island will be scanned or in other words: We will win the game with this teleport.

So how to reduce the actual number of tiles that need to be scanned?

As observed the scan actions of the agent will not bring much benefit in scanning the entire map. Except for the mountain and the sea, the only way to reduce the actual number of tiles that need to be scanned is through the hint.

Assume that a hint provided will have a 50% chance of being true.
We know that the mask generated by hinting when True and when False will cover all the map.
So when verifying a hint, the number of expected cells identified without a treasure is 50% of the map.
From that, verifying a hint always brings more profit for the agent.

We will verify all necessary hints as soon as possible to reduce the actual number of cells to be scanned.

If there is no hint to verify or a suitable location to teleport, what do we do?

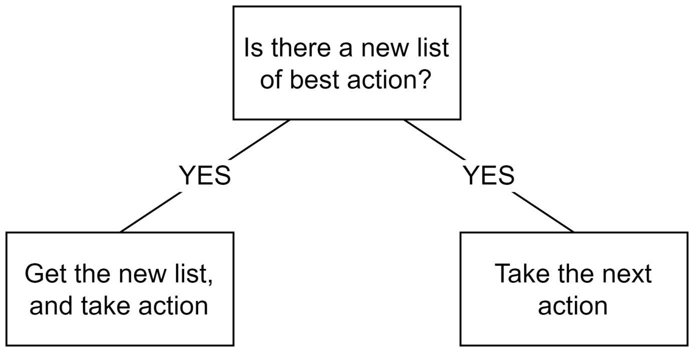
WE SCAN.

We see that conducting a scan is a sequence of actions to move to the next scanning location, if the agent recalculates the sequence of actions each time you want to make a decision, the agent will easily be put into a state of loops back and forth in a few cells.

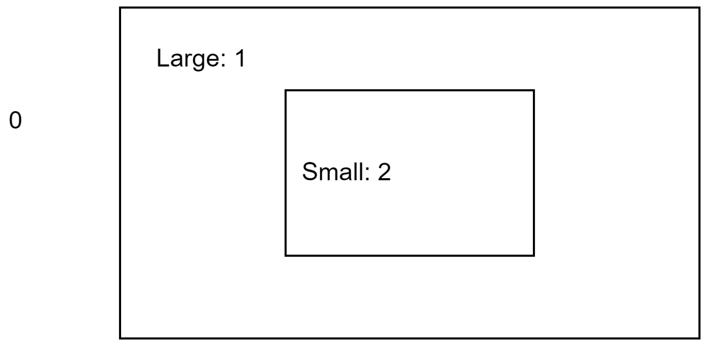
The solution to this is to use a longer-term memory for the agent.

The agent will not change the previous sequence of actions if there is no big change in the map.

The target is the position with the most score and nearest in the last calculation.


Is there a new list of best action?
YES
YES
Get the new list, and take action
Take the next action

We will describe it in more detail here:

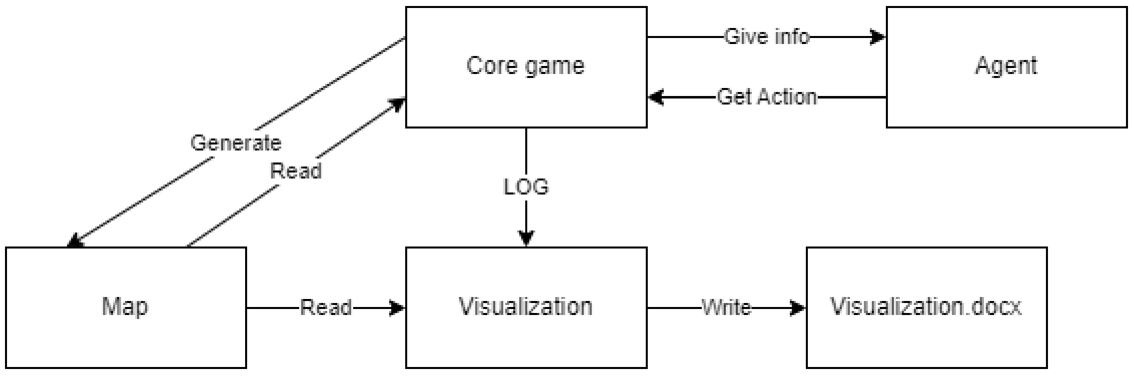


The score for the position = (Number of tile that not know in small range)x 2 + (Number of tile that not know in large range but not in small range)
Small scans can be performed while moving, but large scans require a separate action.

```
**If** *there is no more action in the list to do* **OR**
    *the score of the target = 0* **OR**
    there is the new target that have more score
**Then** Get the new list of action

Take the action from the list.
```

The first two conditions are easy to understand.
The third condition is used to avoid the loop of the agent we mentioned above.
The shortest path to a location is found by BFS with actions 2 and 3 with cost = 1.

III. <u>Pseudo code and Explanation</u>



There are the following files:
- main.py: The core to implement game rules.
- AgentP.py: The agent.
- MapGenerator.py: Generate a map.
- Visualization.py: Visualize the game to docx.
- MapPaint.py: Help Visualization.py to draw the map.
- Map.txt: Store the map for the game.
- test.png: The image drawn by MapPaint.py .
- log.txt: The steps in the game.
- Visualization.docx: same with log.txt but has a colorful map to know what happened.

main.py: Core of the game

We have 5 main stages:

- **Create a map or use another (optional)**: created by MapGenerator.py, this is the bonus so we will explain the details in another part.
- **Read the map**:
  - The first line contains 2 numbers to represent the size of the map (W and H).
  - The second line is the turn number that the pirate reveals the prison (r).
  - The third line is the turn number that the pirate is free and start running to the treasure. (f).
  - The fourth line is the number of regions on the map (including the sea) (R).
  - The fifth line contains 2 numbers to represent the treasure position (Tx, Ty)
  - The next ***H*** lines are the ***H*** rows of a map. Each row contains ***W*** columns that is separated using ";" character. Each cell represents a tile, the format for a cell is ***<spaces><region><tile_type>***, where $0 \leq$ ***<region>*** $<$ R and ***<tile_type>*** = *"M" or "P" if* ***<region>*** $\neq 0$. For example: 0", "4", " 4", "   5", "4P", "   4M".
- **Start the game**: Details below:

```
startGame(): start the game play
    Find the random position for agent
    Agent <-Create with position and Map information
    pirateFree: The turn pirate is free
    prisonReveal: The turn prison is revealed
    For i in pirateFree
        If i == prisonReveal then
            Agent <- prison position

        If first hint:
            Agent <- CreateHint(True)
        Else:
            Agent <- CreateHint(True/False)
        # Agent action
        Turn = 2
        While Turn > 0:
            # Tele don't cost an action
            Turn -= getAction(agent.makeMove())
        If the Agent win Stop
    # The pirate is free, because pirate run two step
    len = (Shortest path from pirate to treasure using Dijkstra + 1) // 2

    For i in len
        Same with the above loop

    If the Agent not win yet then Agent is Lose
```

```
CreateHint(isTrue): return a hint for
    random choice the type of hint
    if isTrue:
        return a hint
    else:
        return a false hint
```

Explanation: The occurrence rate of hints is the same, we think that classification of rare hints is unnecessary.
The hints will all exist in a state that provides information of >= 50% of the map, so on average these hints are the same, and because there are many hints that are not as strong as True/False hints or the special type if it occurs a second time will be useless.

```
getAction(agentAct: agent action): do the action of the agent
    If is the scan action:
        If scan to area of Treasure Then agent win
        Else Lose
    If is the teleport
        return cost = 0
    else
        return cost = 1
```

- **Output to log**: print to Log.txt

```
Output(Log): output to log
    Log: list
    Log <- "is the agent win or not"
    print "number of line in Log" to Log.txt
    print "is the agent win or not" to Log.txt
    for u in Log
        print u to Log.txt
```

- **Visualization**: Instead of raw reading from the Log.txt file, we will pass the list of logs directly from main to Visualization. The log file is also printed to Log.txt with no changes so there will be no difference or deception at all. We will explain in detail in different parts.

MapGenerator.py: Generate a map. (BONUS)

```
MapGenerator(N: Size of the map): Generate the map and other information for the game
    # Map has size N x N with N >= 16
    regionMap: Map store the region
    specialMap: Map store Mountain, Prison, Treasure

    createLand(): land and sea
    createMountain(): mountain
```

```
        createPrisonTreasure(): treasure and prison
```

To make it realistic, we will start from a plot of land in the middle of the map and then spread it out with the ratio of creating the ground. The same region needs to be next together.
If there is no Mountain in the game, any two tiles have a path to each other.

```
createLand(): Create region and sea
        # The further away from the original cell, the lower the rate of ground formation
        rateDown = 1 - (64/N)*2/100
        # Number or different region except 0
        cLim = 20 - 128//N
        Queue: The queue prioritizes cells belonging to a region with a small number of cells.
        Queue <- center of map
        While Queue:
                u <- Queue.pop()
                w <- rate to create a ground
                For v in "The cell next to u"
                If random() <= w:
                        #Create ground at v
                        v <- w*rateDown
                        v <- region of u
                        # Change to other region
                        If random() <= special rate
                                v <- new region (cLim)
                        Queue <- v
```

To make things more consistent the mountain ranges need to be contiguous instead of completely sparse.

```
createMountain():  create mountain
        queue <- 0.05 * Number of cells in Map
        while queue:
                # from a tile that slithers around at a rate of 0.2 to create more mountains
                u <- queue.pop()
                specialMap[u[x]][u[y]] = 'M'
                for v in "Tile next to u"
                        if random() <= 0.2
                                queue <- v
```

At the step of adding prison and treasure, we need to take care that this game might end. An important factor in ending the game is whether the Agent scans for the treasure or the Pirate gets to the treasure. Since the element that Agent can scan the

treasure is out of control, we will focus on making sure the Pirate can get to the treasure.
To ensure this factor, we will proceed to put treasure and a number of prisons on the map. Then we will BFS from the treasure to the prison. If there exists a prison where the treasure cannot reach, we will proceed to make the way to the prison through the removal of mountains.

```
createPrisonTreasure(): Create prison and treasure and make sure that the game
can END
    #Number of prison
    pN = 2 +  int(N/64 * 12)
    #Number of treasure
    tN = 1

    prisonList <- 0.1 * Number of tile in Map
    prisonList <- pN tiles in prisonList
    treausre <- a random tile

    For u in prisonList
        speicalMap[u[x]][u[y]] = 'P'

    speicalMap[treausre[x]][treausre[y]] = 'T'

    # list of prison that can't go to treasure
    lstPrison <- checkIsReachable(treasure, prisonList) # By BFS
    For u in lstPrison
        createPathToTreasure(u, treasure)
```

Regarding the path to the treasure, we don't want to create the shortest path to the treasure because that way the game won't be difficult anymore. We have made the path to the treasure using the Dijkstra algorithm with the condition that a normal cell will have cost = 1 and Mountain will have cost = 10. So the algorithm will try to destroy fewer mountains to reach treasure.
Visualization.py: Visualize the game to docx.

```
Visualization(Log: from main.py): Visual the game step
    # Output to docx "Visualization.docx"
    document: docx
    For u in Log:
        document <- u
        If u is agent step:
            Visual the agent action
        If u is the hint:
            Visual the hint mask
```

```
                    If u is main Mask of the agent
                              Visual the main mask
```

MapPaint.py: Help Visualization.py to draw the map.
Used to redraw the map in a more colorful way

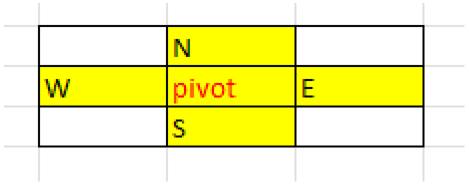
```
MapPaint(map information): draw map
          regionMap: Map store the region
          specialMap: Map store Mountain, Prison, Treasure, Agent
          boundaryMap: store the cells of interest, mark by hint

          For i in range(N)
                    For j in range(M)
                              draw regionMap[i][j]
                              draw specialMap[i][j]
                              draw boundaryMap[i][j]
```

Map.txt: Store the map for the game.(Save map information and other information)
test.png: The image drawn by MapPaint.py (Temporarily save the image to put in the docx file)
log.txt: The steps in the game (Save information about the game's running process)
Visualization.docx: same with log.txt but has a colorful map to know what happend.

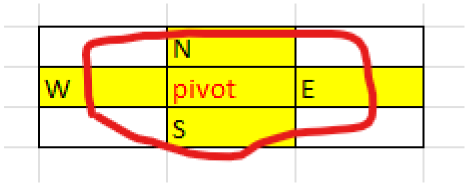
IV. Define game rule

To avoid unnecessary cases, we will clarify the definitions in the program.

- 2 cells that can reach each other when there are no obstacles are two adjacent tiles. The direction is the same with the image below.



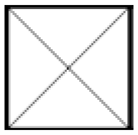
- Boundary of a cell will be its 8 surrounding cells.



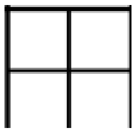
- Agents can't reach cells with mountains, seas, prisons or outside the map size.
- Large scan: 7x7.
- Small scan: 5x5.
- Agent does not know if the cell he is standing in has treasure or not if he does not scan.

How the components are shown in the map:

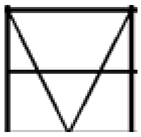
- Mountain: X



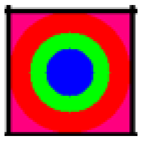
- Prison: +



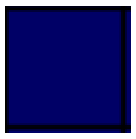
- Agent: V



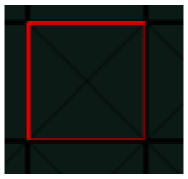
- Treasure: O



- Sea: blue

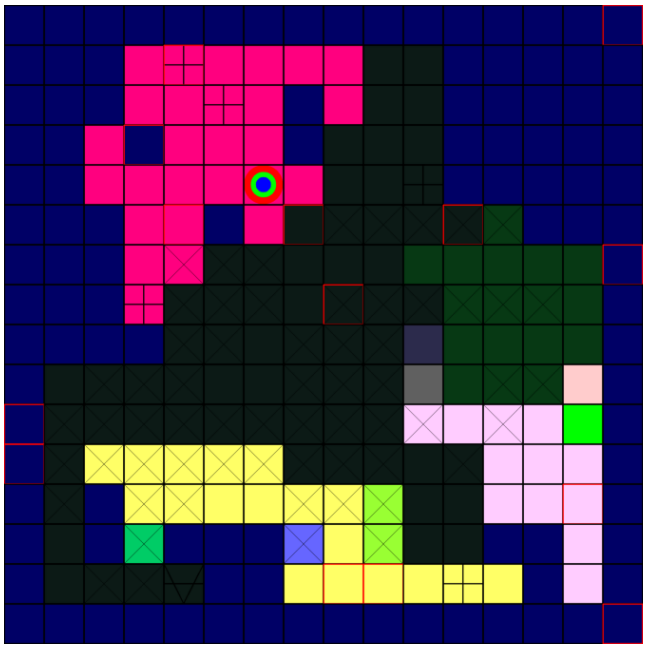


- Boundary: red line

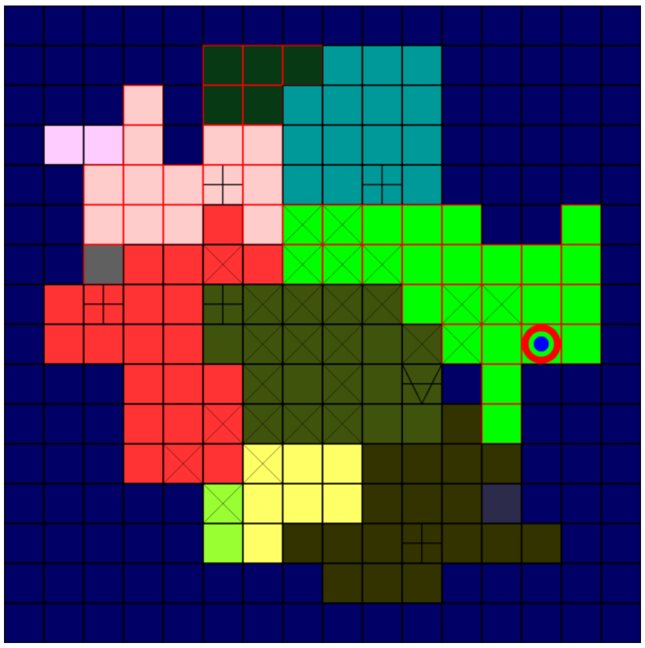


The pirate's hints:

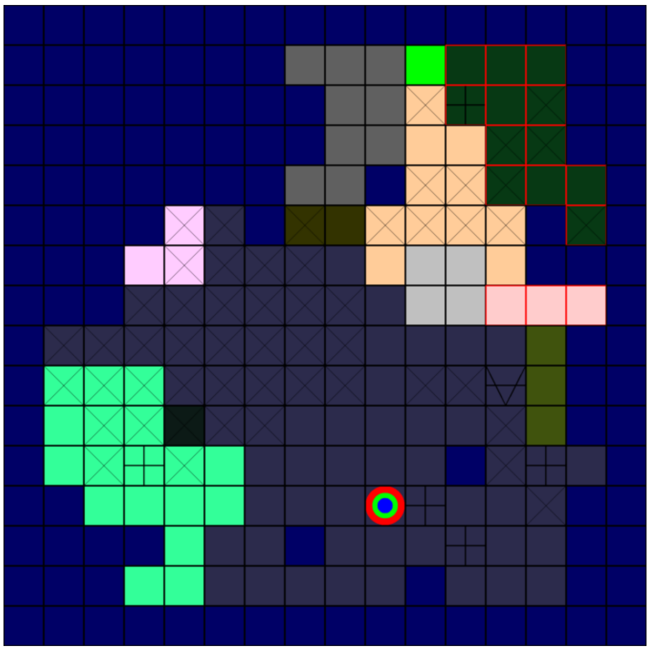
1) A list of random tiles that doesn't contain the treasure (1 to 12).



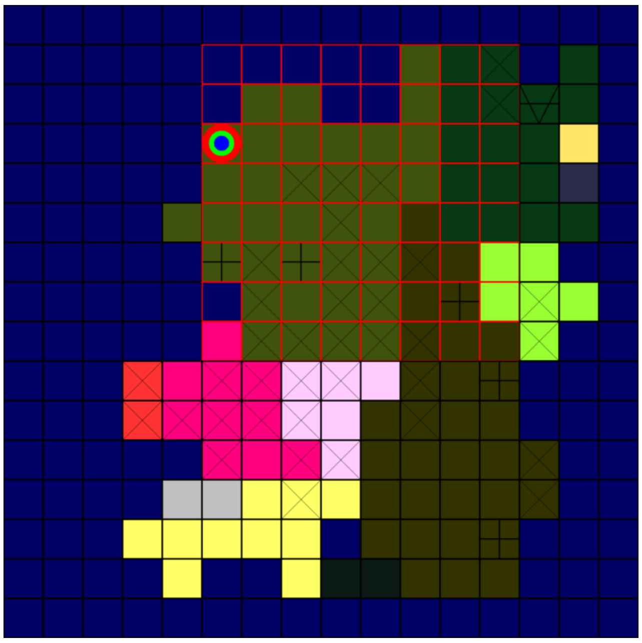
2) 2-5 regions that 1 of them has the treasure.



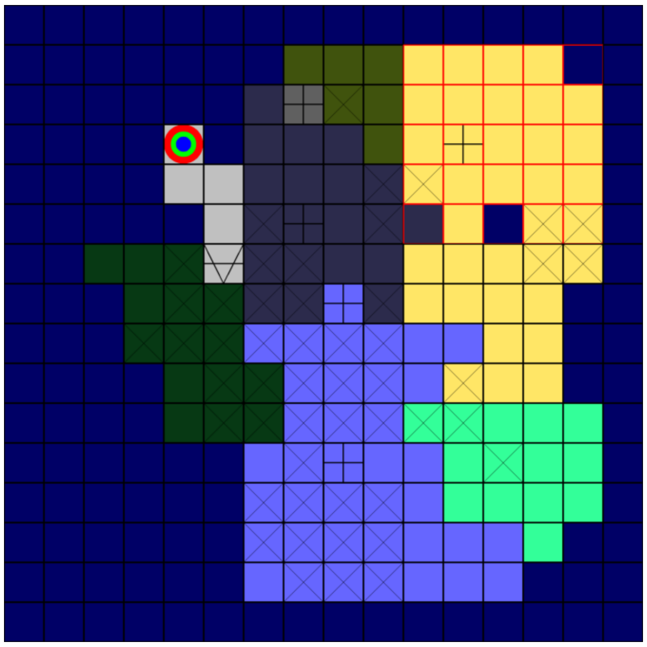
3) 1-3 regions that do not contain the treasure.



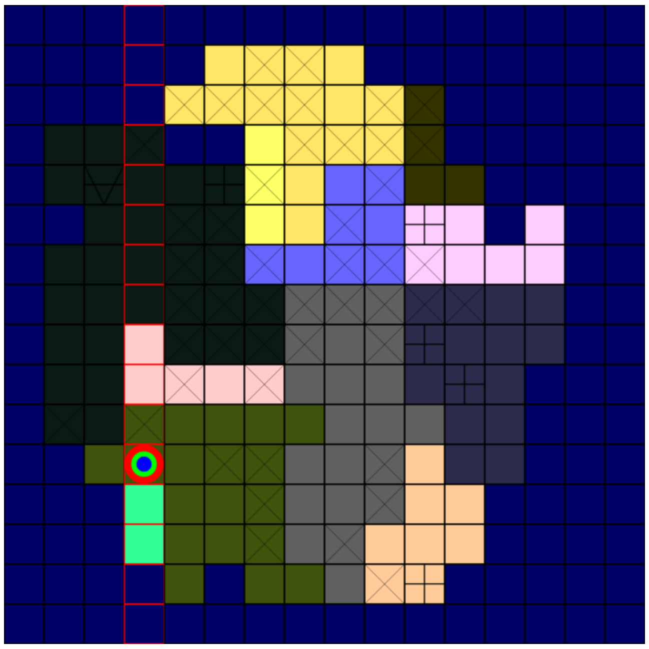
4) A large rectangle area that has the treasure.
8x8



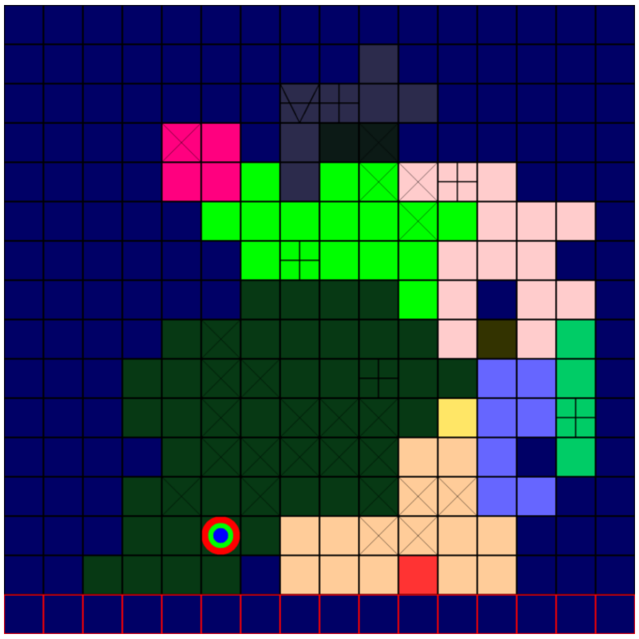
5) A small rectangle area that doesn't have treasure.
5x5



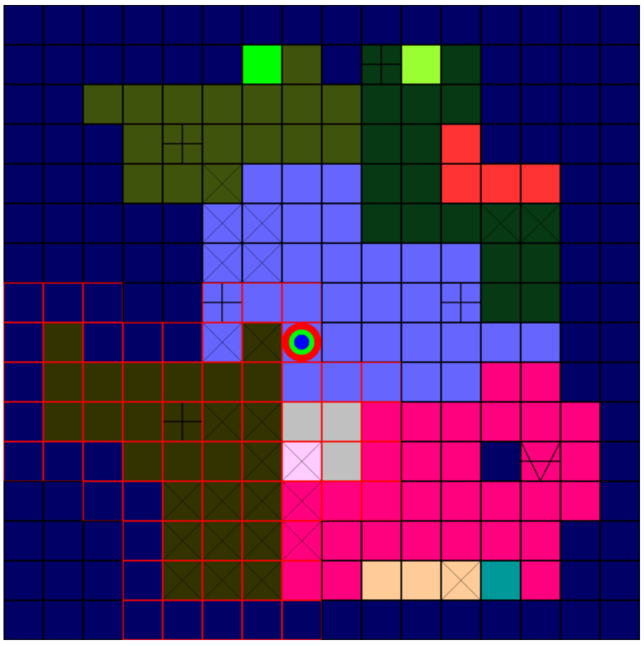
6) He tells you that you are the nearest person to the treasure (between you and the prison he is staying).
We assume the Agent is closer to the treasure than the Pirate's Prison, so cells with a distance from Agent >= to Prison will have no treasure.
This is a special hint that requires Prison information to create a mask.
distance = manhattan distance
7) A column and/or a row that contains the treasure.



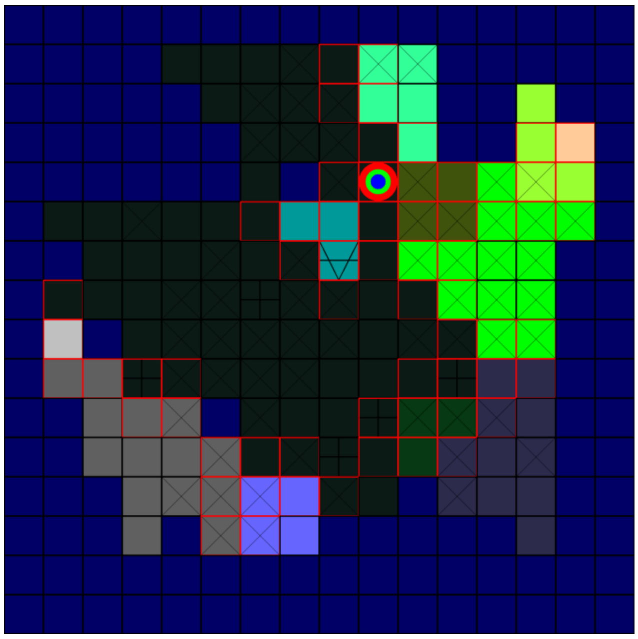
8) A column and/or a row that does not contain the treasure.



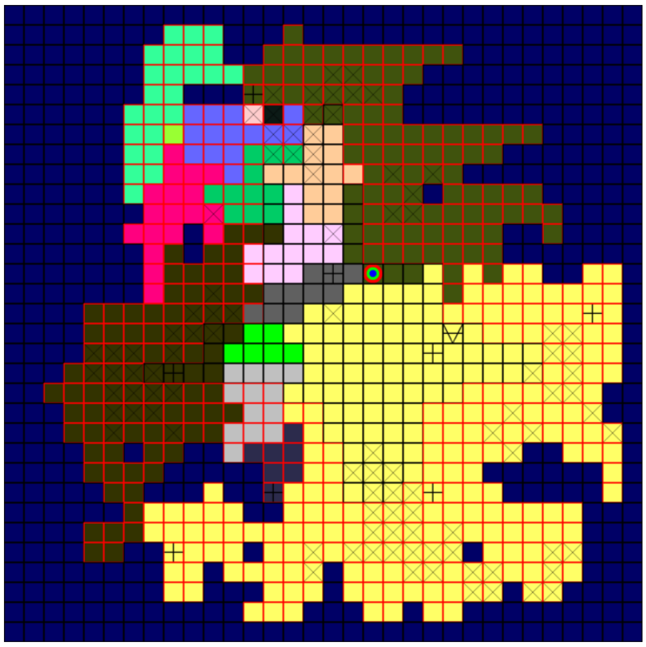
9) 2 regions where the treasure is somewhere in their boundary.
We will redefine this section a bit, it will change to 2 regions that can have treasures at the boundary and within that region. Sea is also a region. And 2 regions next to other/



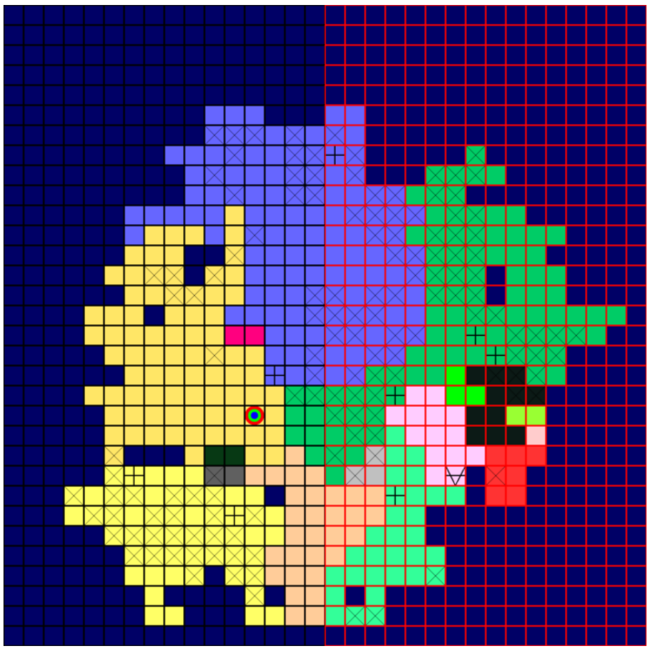
10) The treasure is somewhere in a boundary of 2 regions.
We will redefine this section a bit, it will change to 2 cells with adjacent edges but in 2 different regions (not including sea).



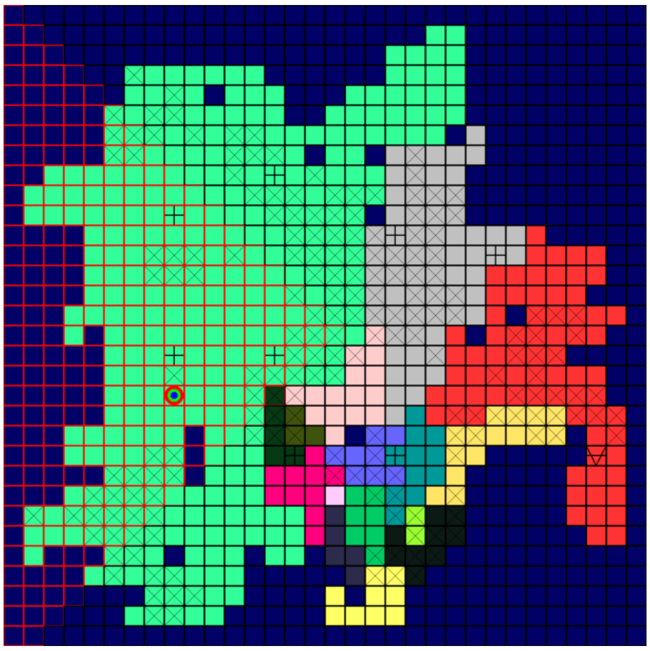
11) The treasure is somewhere in an area bounded by 2-3 tiles from the sea. Cells with sea in their boundary 3 cells around



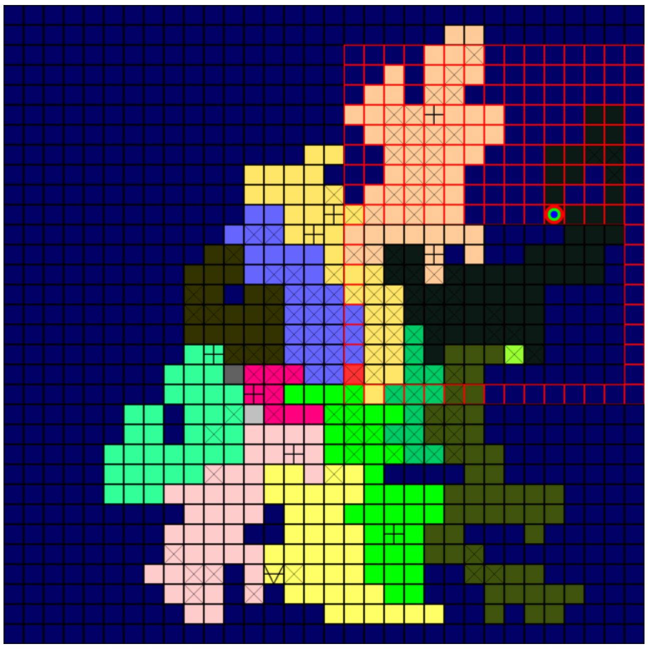
12) A half of the map without treasure.



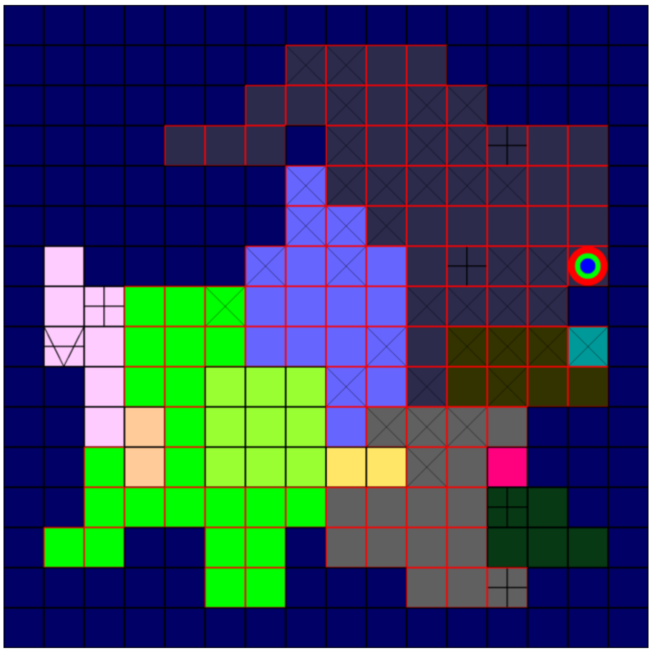
13) From the center of the map/from the prison that he's staying, he tells you a direction that has the treasure (W, E, N, S or SE, SW, NE, NW) (The shape of area when the hints are either W, E, N or S is triangle).



14) 2 rectangles that are different in size, the small one is placed inside the bigger one, the treasure is somewhere inside the gap between 2 rectangles .



15.The treasure is in a region that has mountains.



V. Visualization

In the file visualization the following structure:

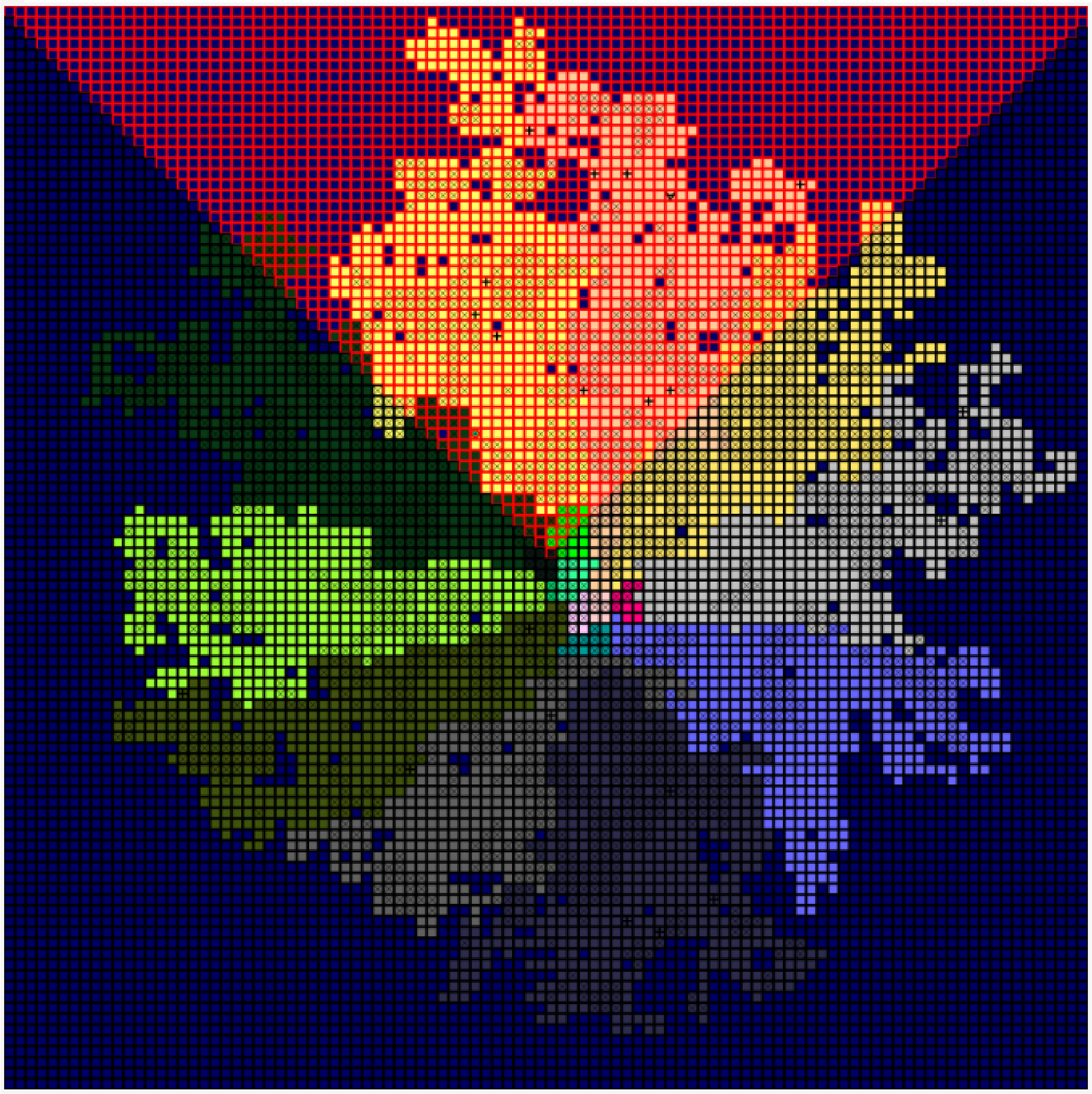
First 2 lines:

```
START
AGENT: x y (position of the agent when start)
```

The next is about each turn.

Each turn will have:

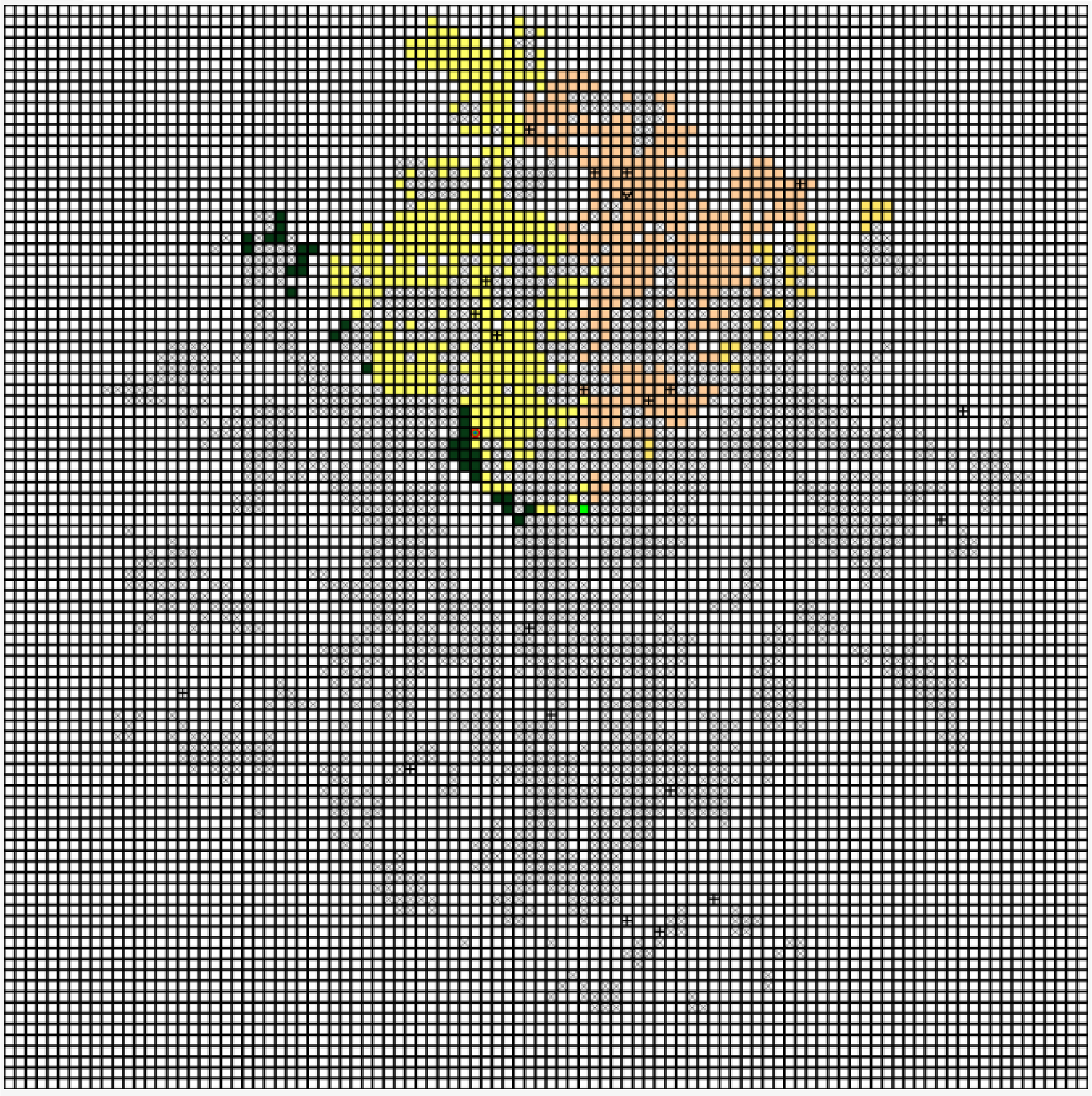
TURN X****** (X: index)
HINT X (X: index)
Type of hint
Information of the hint
The Map of that hint
Ex:
TURN 1**
HINT 1
13
50 50
N



Next there will be 2 types of information Mask and AGENT

Mask

X Y (position of Agent)
The Map Mask
White tile mean scanned or know that nothing in that tile.
Ex:
MASK
17 57



AGENT

AGENT
action description (type ...)
action information
Ex:

```
AGENT
3 MOVE 4 steps, direction S
[1, 0]
4
Ex:
AGENT
VERIFY
2
False
```

At the end is the status of the game

Ex:

WIN

LOSE

VI. Test cases

Pirate reveals the position after [2, 4] turns and Free in turn [x+1, 5] where x is the turn the prison is revealed.

5 maps:

- Map1.txt:
  - Size: 16x16
  - Pirate reveal: 4
  - Pirate Free:
- Map2.txt:
  - Size: 32x32
  - Pirate reveal: 3
  - Pirate Free: 5
- Map3.txt:
  - Size: 64x64
  - Pirate reveal: 3
  - Pirate Free: 4
- Map4.txt:
  - Size: 80x80
  - Pirate reveal: 2
  - Pirate Free: 5
- Map5.txt:
  - Size: 100x100
  - Pirate reveal: 4
  - Pirate Free: 5

VII. Evaluation

We have a number of turns for the agent to win or lose. RED is Lost

Test	**1**	**2**	**3**	**4**	**5**	**6**
Map1.txt	**3**	**1**	**7**	**5**	**6**	**5**
Map2.txt	**10**	**4**	**1**	**2**	**13**	**9**
Map3.txt	**4**	**7**	**9**	**6**	**12**	**6**
Map4.txt	**4**	**6**	**19**	**8**	**11**	**7**
Map5.txt	**11**	**7**	**4**	**15**	**3**	**7**

We see that the agent wins after an average of 7 turns.
To explain the above, besides having luck in the distance, it is largely due to the verify hint.
As mentioned above, We have the expected value when verifying a hint will create a mask covering 50% of the map. In other words, a title with 50% will know if there is a treasure from a hint. So after an average of 7 turns, there is only a 0.5^7 =~0.0078 percent chance of not knowing where the treasure is.
But it should be clarified that this is only the expected value.

Analyze the cause of Lost.
There are 2 main reasons:
- There are too many boxes to scan but we can't scan them all, the pirate has arrived.
- It's almost the same as cause one but here what hinders us is the terrain when the Agent can't move away from a certain area.
Further analysis of being obstructed by terrain. We learn that in the map there will be areas divided by mountains and can only go around in that area.

- If the number of areas is **1**: We lose because we can't scan all the cells.
- If the number of areas is **2**: We have a problem when the Agent and the Treasure are not in the same area so that the Agent can go and scan.

The solution is that we can jump to another area to continue scanning after scanning the old area by teleporting.
But as mentioned beforehand, teleporting is critical and if we fail, we may have to pay the price because we no longer have the tools to win.

- If the number of areas is greater than **>2**: In this case, we almost have to depend on hints to keep the number of cases to the lowest level.

Suggested solutions:

- We propose a method that is to find a sequence of paths such that the number of steps is minimal to scan all remaining cells. This solution is difficult to implement because there can be a continuous loop that causes the agent to only go back and forth in one place. It is similar to the TSP traveling salesman problem but continuously changes the environment state every turn.

- There is also one thing that we have not taken advantage of, which is the information of pirates when free. We can easily estimate the locations that the pirate is in to give the Agent's judgment using the BFS method from the turn-based pirate's prison.

VIII. Conclusion

The results above show that our Agent was able to win most of the cases with an average number of rounds of 7 which is relatively low compared to the time Pirate exited. The results also show that the actual size of the map to be scanned will decrease very quickly, so even if it is large, it will not affect too much.