This code is designed to create a game which is called Radegast's game. The code is written with object-oriented programming in Java. The code consists of two classes and the main class for the application. Two classes are the lake class and the pattern class.

The pattern class is mainly created for the board's (matrix) attributes and its methods. There are lots of methods to update, create and print the board. The board is a matrix and it has "row times column" numbers. When the code prints the board, it has to use scientific notations for rows and columns, so the pattern class has scientific notation numbers and alphabet array lists. Also, there is an array list for scientific notations of lakes which will be determined with methods in lake class. The unified version of scientific notations is stored in an array list and it aims to check whether the input of the location is true or not. The class keeps the max layer number for the loop in the main class which is created to determine lakes inside of it. Most of the methods create some part of the board with loops. The board updates according to the input from a user. The more important class is the lake class which includes the crucial algorithms for this project.

The lake class is mainly created to determine lakes and form their features. The main aim of the project is to determine lakes' locations and their volumes, so the lake class has a 2D volume array, an array list for locations of lakes, and an array list for locations of adjacent volumes. This class has 3 methods. The first of them just creates a volume array of the lake. The other two methods are doing the most critical job in this project. They are created with a breadth-first search algorithm. Firstly, the code in the main finds the lakes in one layer, and with the last method, it finds lakes with all layers. It returns their volumes' matrix which is a significant array list for the remaining part.

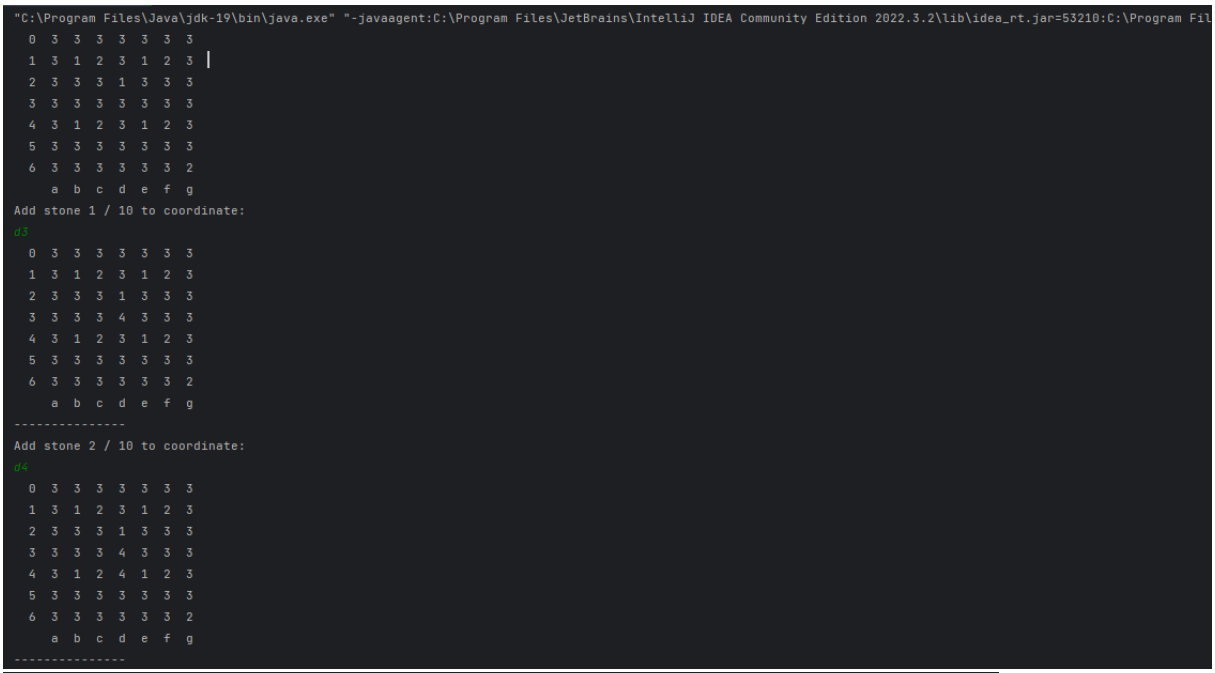
These two methods are created with the same logic. The first one has a parameter which is the matrix layer. It is a matrix with ones and zeros. The first row and column numbers are stored in variables. The "visited" boolean array is created with the length of the row and column to mark locations that the code visits. The lakes array list stores all lakes and the method returns it at the end of the method. The nested for loops iterate through each location in the matrix. If the cell is zero and has not been visited, it's part of a new lake. The method adds it to the queue. The queue is the array list of int arrays. The bfs algorithm works by adding the current cell to the queue if the conditions are met. Then, it marks this cell as visited and explores its neighbors in the directions that are defined earlier (diagonals,up-down,left-right). If a neighbor is a 0 and has not been visited yet, it is added to the queue and marked as visited. The algorithm continues until all cells in the lake have been visited and added to the queue when conditions are true.

As the bfs algorithm has finished, the method checks if any cells in the lake are on the boundary of the matrix because it means that the lake can flow out and so there is no lake in there. If the boolean which checks whether is there an exit or not is false, this lake is added to the lakes array list. Finally, the method returns the lakes array list.

The second method which finds adjacent volumes has the same logic, but this time it checks the volumes matrix of the lakes. It means that it is not just one layer of the lake. It is a lake of a 3D shape. If the volume is zero, it means that there is no lake in this location. If the cell is zero, it controls its neighbor cells and other things are the same as the bfs algorithm. Also, this time it does not check boundaries, because the volumes are already created with the conditions of lakes. At the end of the method, it returns to an array list of lakes' locations.

The main class is created with these two classes. The first pattern class is constructed. The scanner is created to take the matrix from an input text file. With a while loop the matrix is stored with its attributes and the input file scanner is closed. After that, the user has 10 modifications right to alter the board. With while loop inputs are taken and if the input is valid, the board is changed and printed again and again until the alteration number reaches ten. After this process ended, the algorithm part which calculates lakes starts. The lakes are stored in the lakes array list. With for loop, the lakes are calculated until the condition reaches the max layer number. The attributes of lakes are defined with methods of lake class. Then, the final board is created with the method of the final board. At the end of the code, the final board is printed, and the final score which is the sum of the square root of the volumes of lakes.

Example 1:



```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.3.2\lib\idea_rt.jar=53210:C:\Program Fil
  0  3  3  3  3  3  3  3
  1  3  1  2  3  1  2  3
  2  3  3  3  1  3  3  3
  3  3  3  3  3  3  3  3
  4  3  1  2  3  1  2  3
  5  3  3  3  3  3  3  3
  6  3  3  3  3  3  3  2
     a  b  c  d  e  f  g
Add stone 1 / 10 to coordinate:
d3
  0  3  3  3  3  3  3  3
  1  3  1  2  3  1  2  3
  2  3  3  3  1  3  3  3
  3  3  3  3  4  3  3  3
  4  3  1  2  3  1  2  3
  5  3  3  3  3  3  3  3
  6  3  3  3  3  3  3  2
     a  b  c  d  e  f  g
---------------
Add stone 2 / 10 to coordinate:
d4
  0  3  3  3  3  3  3  3
  1  3  1  2  3  1  2  3
  2  3  3  3  1  3  3  3
  3  3  3  3  4  3  3  3
  4  3  1  2  4  1  2  3
  5  3  3  3  3  3  3  3
  6  3  3  3  3  3  3  2
     a  b  c  d  e  f  g
---------------
```

```
Add stone 3 / 10 to coordinate:
d5
  0  3  3  3  3  3  3  3
  1  3  1  2  3  1  2  3
  2  3  3  3  1  3  3  3
  3  3  3  3  4  3  3  3
  4  3  1  2  4  1  2  3
  5  3  3  3  4  3  3  3
  6  3  3  3  3  3  3  2
     a  b  c  d  e  f  g
---------------
Add stone 4 / 10 to coordinate:
e5
  0  3  3  3  3  3  3  3
  1  3  1  2  3  1  2  3
  2  3  3  3  1  3  3  3
  3  3  3  3  4  3  3  3
  4  3  1  2  4  1  2  3
  5  3  3  3  4  4  3  3
  6  3  3  3  3  3  3  2
     a  b  c  d  e  f  g
---------------
Add stone 5 / 10 to coordinate:
f5
  0  3  3  3  3  3  3  3
  1  3  1  2  3  1  2  3
  2  3  3  3  1  3  3  3
  3  3  3  3  4  3  3  3
  4  3  1  2  4  1  2  3
  5  3  3  3  4  4  4  3
  6  3  3  3  3  3  3  2
     a  b  c  d  e  f  g
---------------
```

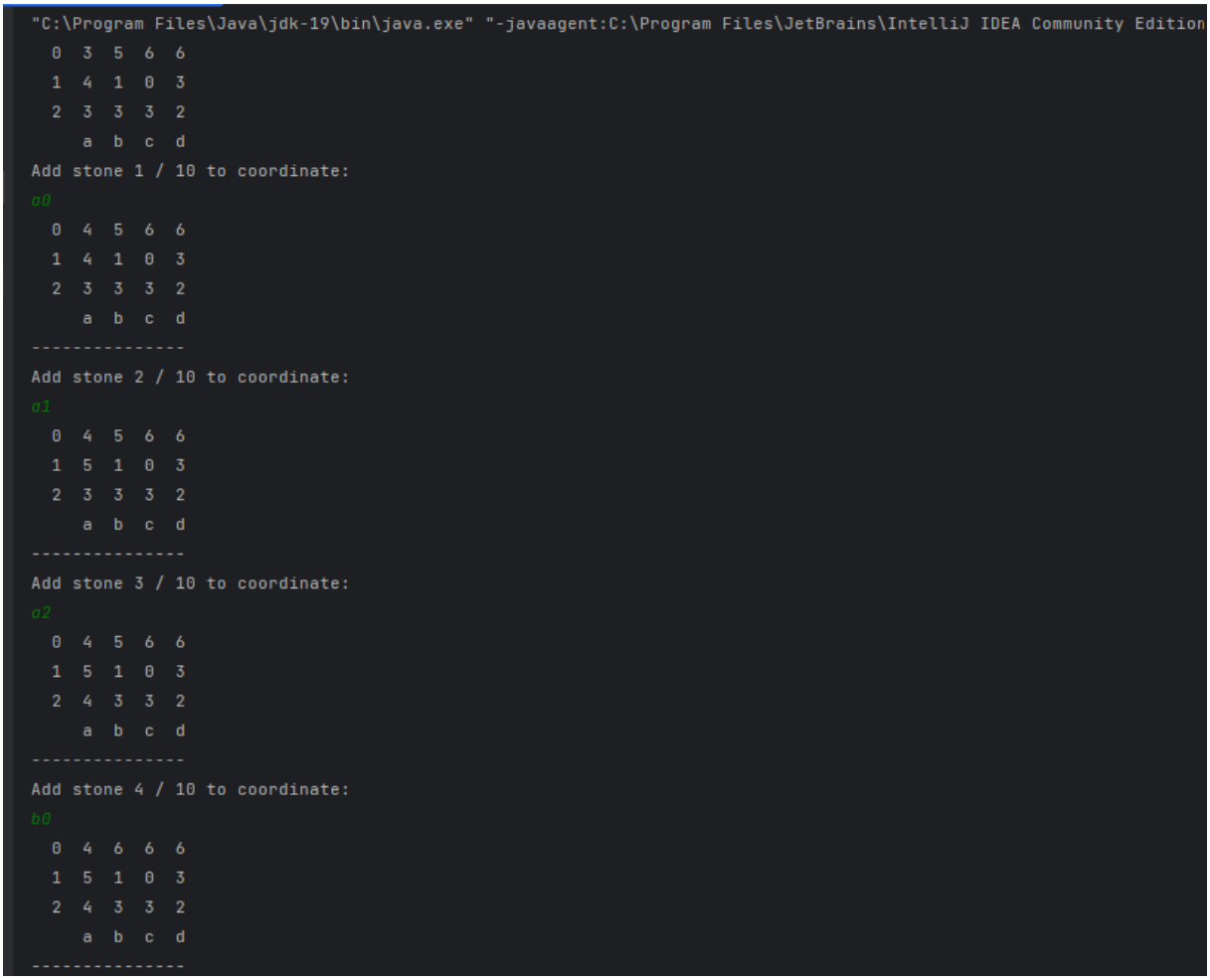
```
Add stone 6 / 10 to coordinate:
g5
  0  3  3  3  3  3  3  3
  1  3  1  2  3  1  2  3
  2  3  3  3  1  3  3  3
  3  3  3  3  4  3  3  3
  4  3  1  2  4  1  2  3
  5  3  3  3  4  4  4  4
  6  3  3  3  3  3  3  2
     a  b  c  d  e  f  g
---------------
Add stone 7 / 10 to coordinate:
g4
  0  3  3  3  3  3  3  3
  1  3  1  2  3  1  2  3
  2  3  3  3  1  3  3  3
  3  3  3  3  4  3  3  3
  4  3  1  2  4  1  2  4
  5  3  3  3  4  4  4  4
  6  3  3  3  3  3  3  2
     a  b  c  d  e  f  g
---------------
Add stone 8 / 10 to coordinate:
g3
  0  3  3  3  3  3  3  3
  1  3  1  2  3  1  2  3
  2  3  3  3  1  3  3  3
  3  3  3  3  4  3  3  4
  4  3  1  2  4  1  2  4
  5  3  3  3  4  4  4  4
  6  3  3  3  3  3  3  2
     a  b  c  d  e  f  g
---------------
```

```
Add stone 9 / 10 to coordinate:
f3
  0  3  3  3  3  3  3  3
  1  3  1  2  3  1  2  3
  2  3  3  3  1  3  3  3
  3  3  3  3  4  3  4  4
  4  3  1  2  4  1  2  4
  5  3  3  3  4  4  4  4
  6  3  3  3  3  3  3  2
     a  b  c  d  e  f  g
---------------
Add stone 10 / 10 to coordinate:
e3
  0  3  3  3  3  3  3  3
  1  3  1  2  3  1  2  3
  2  3  3  3  1  3  3  3
  3  3  3  3  4  4  4  4
  4  3  1  2  4  1  2  4
  5  3  3  3  4  4  4  4
  6  3  3  3  3  3  3  2
     a  b  c  d  e  f  g
---------------
  0  3  3  3  3  3  3  3
  1  3  A  A  3  A  A  3
  2  3  3  3  A  3  3  3
  3  3  3  3  4  4  4  4
  4  3  B  B  4  C  C  4
  5  3  3  3  4  4  4  4
  6  3  3  3  3  3  3  2
     a  b  c  d  e  f  g
Final score:6,80
Process finished with exit code 0
```

Example 2:

https://drive.google.com/drive/folders/1Rw6BEqoBthdjhnZpCEPCt-D3C4iKM5Po?usp=share_link

Example 3:



```
Add stone 5 / 10 to coordinate:
b1
  0  4  6  6  6
  1  5  2  0  3
  2  4  3  3  2
     a  b  c  d
---------------
Add stone 6 / 10 to coordinate:
b2
  0  4  6  6  6
  1  5  2  0  3
  2  4  4  3  2
     a  b  c  d
---------------
Add stone 7 / 10 to coordinate:
c0
  0  4  6  7  6
  1  5  2  0  3
  2  4  4  3  2
     a  b  c  d
---------------
Add stone 8 / 10 to coordinate:
c2
  0  4  6  7  6
  1  5  2  0  3
  2  4  4  4  2
     a  b  c  d
---------------
Add stone 9 / 10 to coordinate:
d0
  0  4  6  7  7
  1  5  2  0  3
  2  4  4  4  2
     a  b  c  d
---------------
```

```
Add stone 10 / 10 to coordinate:
d1
  0  4  6  7  7
  1  5  2  0  4
  2  4  4  4  2
     a  b  c  d
---------------
  0  4  6  7  7
  1  5  2  A  4
  2  4  4  4  2
     a  b  c  d
Final score:1,41
Process finished with exit code 0
```

```
Add stone 5 / 10 to coordinate:
b1
  0  4  6  6  6
  1  5  2  0  3
  2  4  3  3  2
     a  b  c  d
---------------
Add stone 6 / 10 to coordinate:
b2
  0  4  6  6  6
  1  5  2  0  3
  2  4  4  3  2
     a  b  c  d
---------------
Add stone 7 / 10 to coordinate:
c0
  0  4  6  7  6
  1  5  2  0  3
  2  4  4  3  2
     a  b  c  d
---------------
Add stone 8 / 10 to coordinate:
c2
  0  4  6  7  6
  1  5  2  0  3
  2  4  4  4  2
     a  b  c  d
---------------
Add stone 9 / 10 to coordinate:
d0
  0  4  6  7  7
  1  5  2  0  3
  2  4  4  4  2
     a  b  c  d
---------------
```