Advances in Engineering Research, volume 119
2nd International Conference on Automatic Control and Information Engineering (ICACIE 2017)

Tile Laying Problem Solving Based on Java Object-Oriented

Lin Chengshi
Information Technology Department
Hainan Vocational College of Political Science and Law
Haikou Hainan, the People's Republic of China
28775834@qq.com

Huang Binwen
Information Technology Center
Hainan Vocational College of Political Science and Law
Haikou Hainan, the People's Republic of China
64471362@qq.com

***Abstract*—Unlike the general tile laying problem, the tiles, in this case, have a variety of different shapes and can be rotated when placed, thus increasing the difficulty of solving the problem. Through the object-oriented analysis method, the article analyses the characteristics of the tiles and flooring, constructs the problem model, and uses the retrospective thought to realize all the solution space in the tile lying, and finally, solve the problem.**

Keywords—tile problem; object-oriented; algorithm

I. INTRODUCTION

In the 6th National Software Competition (Blue Bridge Cup) final there was a test with the laying of tiles. The question is to have a size of n * m room floor, the use of the following two shapes of tile laying [1], as shown in Fig.1, placed when the tile can be rotated. Hence how many layouts are available if to fully cover the whole room floor? For example, when the floor size is 4 * 4, there are two kinds of placement plan, as shown in Fig.2.

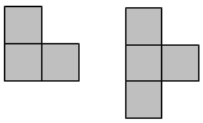

Fig. 1. Two shapes of tiles

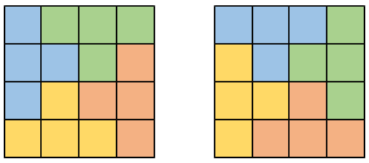

Fig. 2. Floor of the 4 * 4 when the two kinds of laying program

II. THE PROBLEM ANALYSIS

A. Analysis of the Characteristics of Tiles and Floors in the Problem

By observing the tiles can be roughly seen as a rectangle consisting of multiple squares. The box has a solid and hollow two, thus it forms a different shape of the tile. Tiles have four directions, according to a certain order of rotation; it can produce different graphics [2]. As shown in Figure 3.

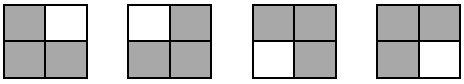

Fig. 3. Tile in four different directions

The floor is a regular n * m rectangular box. For the floor, a basic position (referred to as a base point) is required to lay the tiles in order to determine where the tile is placed. The tiles also need to have a basic point, which is used when the laying of the floor with the base point alignment to determine the other points on the tile placed relative position. The base point of the floor is the point where the unused tiles are to be placed, and it is changed, and each time a tile is laid, it will redefine the next base point, as shown in Figure 4. The basic point of the tile is generally the first solid point (from top to bottom, from left to right) in the series of tiles. As shown in Figure 5.

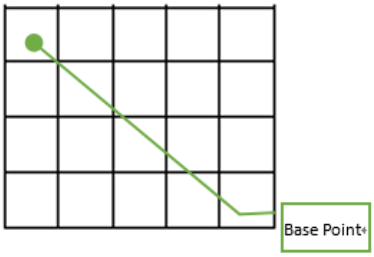

Fig. 4. Base of the floor

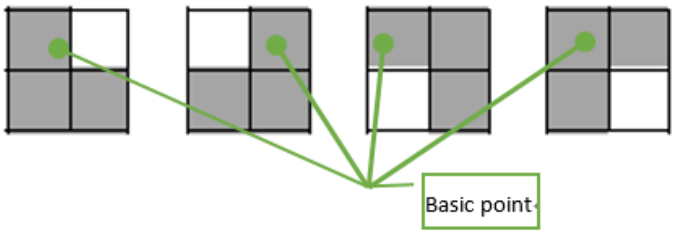

Fig. 5. The basic point of the tile

B. The General Process of Tile Laying to the Floor

The process of laying the tiles on the floor can be seen as a process of grouping, these box points represent the tiles into groups of n * m squares. At the beginning of the floor to the upper left corner of the first point as the starting point, it takes the first direction of the first tile to lay. The basic point of the

tile is placed on the floor with the base point on the floor. The other points on the tile are referenced to the basic point and placed in the relative coordinate position, which cannot exceed the effective range of the floor and cannot coincide with the point where the floor has been placed. If the success of the placement of the corresponding floor on the floor of the state to be marked as occupied, and then it need to check whether the floor has been covered, if covered with the number of layout plus one, if not, then moving the base point to the next new location to repeat the above Laying process. If there is no success, it needs to continue to try the direction of the tile and use another tile.

In order to get all the possible laying options, it is necessary to use the backtracking algorithm to traverse all possible solutions of the laying of tiles. Such as a base point on the floor when there is no laying down the tile, it will return to the previous point using the location of the current tile type corresponding to the next tile shape to re-lay; if all tile types are traversed, but also to return to the previous point to continue this process until the beginning of the base point of all the types of tiles are traversed after the end of the entire laying process. After each floor is covered, it is necessary to take a backtracking algorithm: if the current tile is not the last type of tile, then continue at the current base point of the auxiliary, or still need to return to a base point using the location of the current tile type corresponding to the next A tile shape is re-laid.

C. Object-oriented Analysis of the Problem

The object in this question is relatively simple; there are two different tiles and a floor. Because there are many types of tiles, it is possible to design an abstract parent class to define the characteristics and behavior of the tiles, which is derived from the parent class. Flooring because there is no sub-category, so here is only a design of a floor class can be [3].

III. THE SPECIFIC REALIZATION

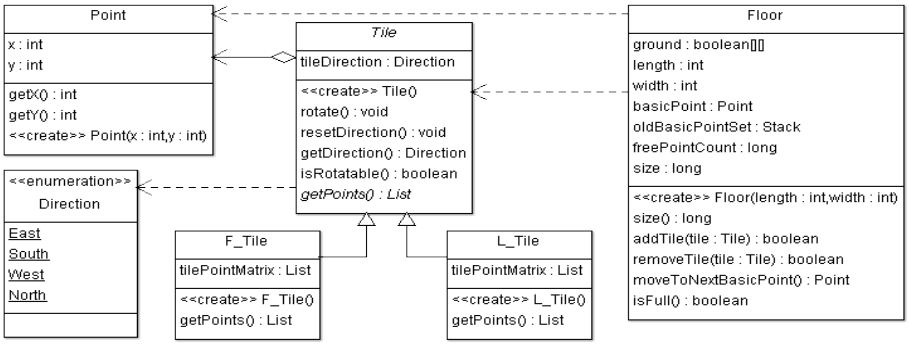

Fig. 6. Class and its relationship diagram

A. Tile Parent Class and Its Subclass Design

According to the previous object-oriented analysis, we firstly design a "tile" abstract class Tile, which has a direction attribute tileDirection and three important methods: rotate, isRotatable, and the getPoints. Where the rotate method is used to change the direction of the tile; the isRotatable method determines whether the tile is rotated; the getPoints method is an abstract method, implemented by different subclasses, and its function is to obtain the coordinates of the points according to the different directions of the tiles.

This abstract class, Tile, derives two different "tile" subclasses, L_Tile and F_Tile, representing two types of tiles in the subject. There is a tile Point Matrix property in the tile class that holds the coordinates of the tiles in different directions. In the coordinates of the different shapes of the tile, you only need to consider the tile position in the solid box can be, where the coordinates are used to represent the box in the tile position. The following is the first "L" shape of the tile, for example, the four coordinates of the coordinates and the corresponding set of the lattice as shown below:

Fig. 7. Correspondence of the coordinates of each point of the tile

The first group of tiles of the coordinates of the series: (0,0), (1,0), (1,1);

The second group of tiles of the coordinates of the series: (0,1), (1,0), (1,1);

The third group of tiles of the coordinates of the series: (0,0), (0,1), (1,1);

The fourth group of tiles of the coordinates of the series: (0,0), (0,1), (1,0).

Specify the top of each tile from the first few points as the basic point of the tile structure. The tile class implements the abstract method getPoints in the parent class Tile, which returns the coordinates of the series according to the direction of the current tile.

The following is the L-shaped tile class code:

```
public class L_Tile extends Tile {
```

```
    private List<Point> tilePointMatrix;// Tile shape dot matrix (dot set)
        // Construction method
        public L_Tile() {
                super();
                tilePointMatrix = new ArrayList<Point>();
        }
        @Override
        public List<Point> getPoints() {
                tilePointMatrix.clear();
                switch (getDirection()) {
                case East:
                tilePointMatrix.add(new Point(0, 0));
                tilePointMatrix.add(new Point(1, 0));
                tilePointMatrix.add(new Point(1, 1));
                        break;
                case South:
                ……//refer to above case
                        break;
                case West:
                        ……//refer to above case
                        break;
                case North:
                        ……//refer to above case
                }
                return tilePointMatrix;
        }
}
```

B. The Design of Flooring

According to the previous analysis, the flooring class has two important attributes: ground and base point. Ground is a logical array that holds (N * M) square information. Square, each cell has two states that false or true, in the laying of tiles when the tiles covered with a true place, there is no bedding with false said, before the start of the floor mattress all false. The floor class has three important ways: the addTile method is used to load the tiles into the floor; the removeTile method removes the tiles from the floor; the isFull method determines whether the floor has been covered by the tiles. In addition, in order to program the floor class, there are other attributes and methods, limited space here is not one by one introduced [4]. The following is part of the implementation of the code:

```
private boolean[][] ground;// The ground, save the state when laying
private Point basicPoint; // Laying the base point refers to the floor from top to bottom from left to right direction of the first point which is not used.
Private Stack<Point> oldBasicPointSet;// Record the location of the base when the laying of tiles in front
private long freePointCount;// Save the number of empty points that are not currently used

    ……// omit
        // Construction method
public Floor(int length, int width) {
    ground = new boolean[length][width]; // Initialize the size of the floor
    basicPoint = new Point(0, 0); // Start at the base of the upper left corner of the floor
    oldBasicPointSet = new Stack<Point>(); // Create a new stack for saving each base point when laying
                ……// omit
        }
    ……//
public boolean addTile(Tile tile) {
    List<Point> pointSet=tile.getPoints();
        // If it is empty or already covered with tiles, it returns false
        if (pointSet.isEmpty() || isFull()) {
                return false;
        }
    // The first point in the collection point collection is used as a reference point
        Point firstPoint = pointSet.get(0);
        int i, j;
//  Check that each position on the tile relative to the first reference point is normally filled in the floor
        for (Point p : pointSet) {
        i = basicPoint.x + (p.x - firstPoint.x);
        j = basicPoint.y + (p.y - firstPoint.y);
// If the coordinates of the cross-border return false
        if (i < 0 || i >= this.length || j < 0 || j >= this.width) {
                                return false;
                        }
// Returns false if the current location is already in use
        if (ground[i][j]) {
                return false;
                        }
                }
// Above all the questions will fill all the points in the floor
        for (Point p : pointSet) {
                i = basicPoint.x + (p.x - firstPoint.x);
                j = basicPoint.y + (p.y - firstPoint.y);
//  In the corresponding position on the floor marked as the use of state
                ground[i][j] = true;
//Number of remaining empty points
                freePointCount--;
                }
// Record the current base point to the base point stack
                oldBasicPointSet.push(basicPoint);
// Move to a new base point
                ……// omit
                return true;
        }
        public boolean removeTile(Tile tile) {
                List<Point> pointSet=tile.getPoints();
        // If it is empty, it returns false
                if (pointSet.isEmpty() || oldBasicPointSet.isEmpty()) {
                        return false;
                }
```

```
// Remove the previous base point from the base stack as the current base point
        basicPoint = oldBasicPointSet.pop();
// The first point in the collection is taken as the reference point
        Point firstPoint = pointSet.get(0);
        int i, j;
// Check whether each position on the tile relative to the first reference point can be removed from the floor
        for (Point p : pointSet) {
        i = basicPoint.x + (p.x - firstPoint.x);
        j = basicPoint.y + (p.y - firstPoint.y);
        // If the coordinates of the cross-border return false
        if (i < 0 || i >= this.length || j < 0 || j >= this.width) {
                return false;
                }
        }
// Above the problem will remove these points from the floor
        for (Point p : pointSet) {
        i = basicPoint.x + (p.x - firstPoint.x);
        j = basicPoint.y + (p.y - firstPoint.y);
// Mark the unused state at the corresponding position on the floor
                ground[i][j] = false;
// The number of remaining empty points is increased by one
                freePointCount++;
        }
        return true;
    }
// Method: Determine if the floor has been laid
    public boolean isFull() {
        if (freePointCount == 0) {
            return true;
        } else {
            return false;
        }
    }
}
```

C. Operational Class Design

The main process of laying tile is the key algorithm to solve this problem. The algorithm flow chart is shown in Figure 8:

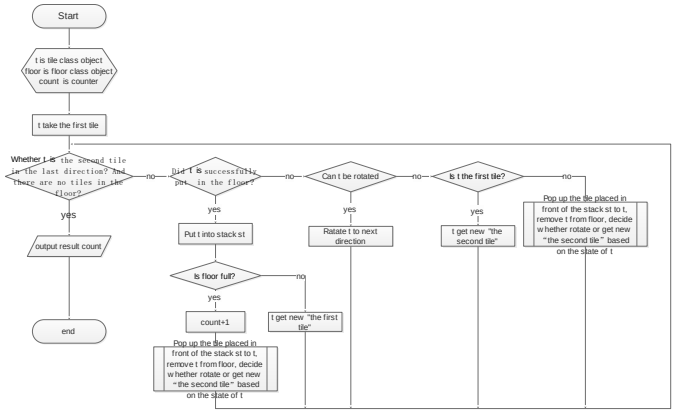

Fig. 8. Flow chart of the operational class algorithm

When the tile is covered with the floor or all the tiles are used in all directions, the backtracking is used to take the tiled tile in front of it, depending on its status. Rotate the new direction or switch to the second tile to continue Tile laying. The flow chart of its backtracking procedure is shown in Fig. 9.

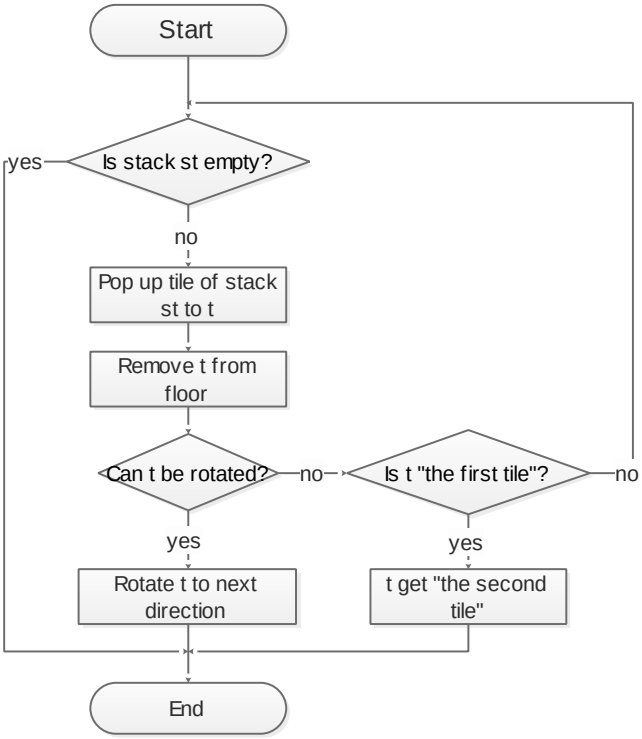

Fig. 9. The flow chart of its backtracking procedure

The main code is as follows:

```
public class Test {
    public static void main(String[] args) {
// Stack - used to record used tiles
        Stack<Tile> st_Tile = new Stack<Tile>();
// Count the number of times the floor was successfully laid
        int count = 0;
// Receive the size of the keyboard input floor
        int length, width;
        Scanner in = new Scanner(System.in);
        length = in.nextInt();
        width = in.nextInt();
        in.close();
// Create a floor object
        Floor floor = new Floor(length, width);
// At the beginning of the first tile
        Tile t = new L_Tile(); // Take out a tile
// Remove all kinds of tiles and lay them into the floor
        while (!(st_Tile.isEmpty() &&
t.getClass().getName().equals("F_Tile") && !(t.isRotatable())))
{
        if (floor.addTile(t)) { //  If the current type of tile can be properly laid to the floor
                st_Tile.push(t); // Add such tiles
```

```
                if (floor.isFull()) { // If the whole floor is
already covered
                count++;// The number of success plus 1
// Check out the previously laid tiles
                while (!st_Tile.isEmpty()) {
                  t = st_Tile.pop(); // Pops up the last floor of
the tile

        floor.removeTile(t); // Remove this tile from the floor
                                                        if
(t.isRotatable()) {// Can you rotate?

        t.rotate();

        break;
                                                        } else if
(t.getClass().getName().equals("L_Tile")) {

        t = new F_Tile();//If it is the first kind of change for
the second tile

        break;
                                                        }
                                                }
                                        } else { // If not covered
                                                t = new L_Tile();
// Then the next new location is still the first tile to lay
                                }
                                } else {// If not properly laid
                                        if (t.isRotatable())
{//Rotate the tiles ready for the next attempt
                                                t.rotate();
                                        } else if
(t.getClass().getName().equals("L_Tile")) {
                                                t = new
F_Tile();// 用另一种瓷砖尝试Try with another tile
                                        } else {// Go back out of
the front tiles
                                                while
(!st_Tile.isEmpty()) {
                                                        t =
st_Tile.pop(); // Pops up the current tiles

        floor.removeTile(t); // Remove this tile from the floor
                                                        if
(t.isRotatable()) {

        t.rotate();

        break;
                                                        } else if
(t.getClass().getName().equals("L_Tile")) {

        t = new F_Tile();

        break;
                                                        }
                                                }
                                        }
                                }
                        }
                        System.out.println("successful completion "
+ count + " laying！ ");
      System.out.println ("successful completion" + count +
"laying!");
        }
```

IV. SUMMARY

In this question, the backtracking depth algorithm is used. When the scale is large, the program takes a long time. For this problem can be added in the backtracking constraints to remove unnecessary branches of the algorithm to optimize. One possible approach is to take into account the structure of the two types of tiles, one consisting of three squares and the other consisting of four squares, so the total number of squares and the number of squares on the floor must be equal to 3x + 4y, where x and y are greater than or equal to 0 integer, so in the backtracking can first check this constraint, the branch does not meet the pruning to achieve the optimization of the algorithm. Of course, there are many ways to optimize the algorithm; interested readers can explore. Tile laying problems can be widely used to solve practical problems such as decorative puzzles, structural joints and other related issues.

REFERENCES

[1] Wang Xiaodong, “Design and analysis of computer algorithms (Third Edition),” Electronics Industry Press, 2007.

[2] Steven S.Skiena, “The Algorithm Design Manual(Second Edition), ” Springer, 2008.

[3] プログラミングコンテストチャレンジブック, “Challenge Programming Contest(Second Edition),” Posts and Telecommunications Press, 2013.

[4] Robert Sedgewick, Kevin Wayne, “Algorithms(Fourth Edition),” Posts and Telecommunications Press, 2012.