跳棋系統分析報告書

1111855 郭祖文
1112012 卓宇恒
1112021 李峻宇
1112024 康承宇
1112043 黃莉淇

一、系統概述

本程式為一個棋盤遊戲，透過繪圖和事件處理實現棋盤和棋子的互動功能。棋盤形狀為每個點為圓形所組成的六邊形和外圈六個八邊形，並配置不同顏色的普通棋子與國王棋子。玩家可點擊棋子並移動到鄰近的空格。程式具備棋子鄰居檢查、顏色變化、棋子位置更新等功能，並透過遊戲規則與勝利條件，例如將國王棋子移動到指定位置即為勝利，並且即可結束遊戲。

二、系統目標

1.提供互動式棋子

設計一個內部六邊形和外圈六個八邊形的圖形化棋盤遊戲，讓玩家能夠透過滑鼠點選棋子並執行遊戲規則。

2.國王棋子勝利條件

確保遊戲邏輯能判斷國王棋子是否移動至指定位置，並正確顯示遊戲結束提示。

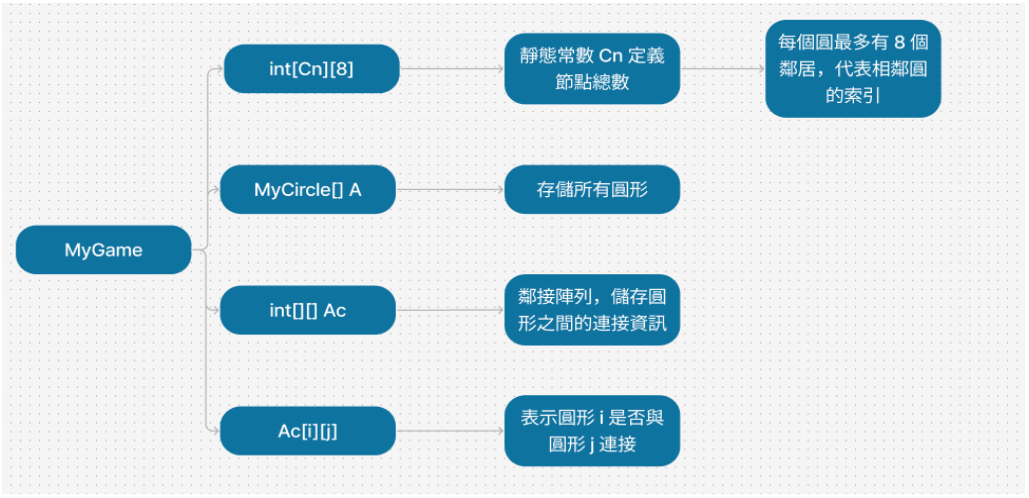
3.鄰接檢查與正確移動判定

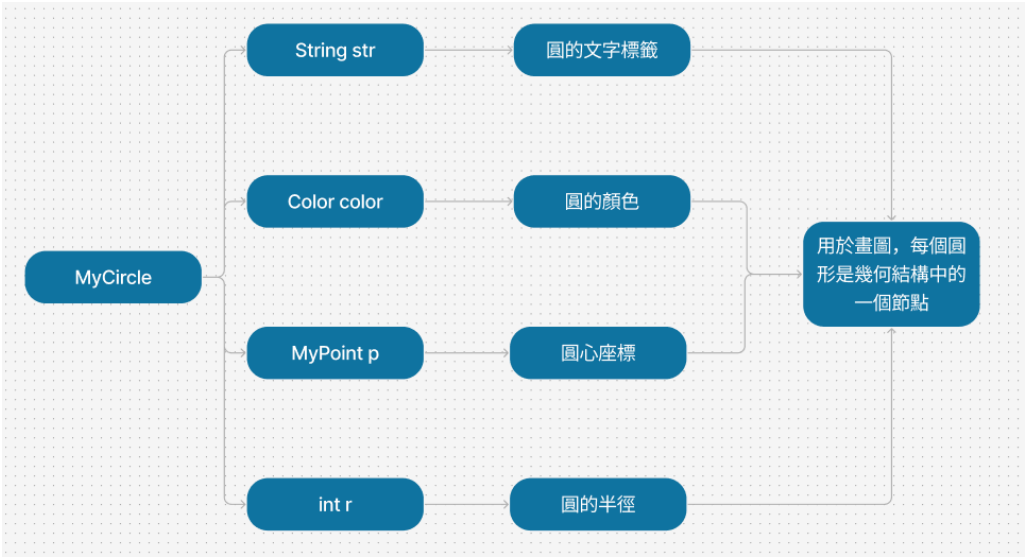
程式系統準確計算棋子的鄰居節點，並限制棋子只能移動至合法的相鄰一格空格。

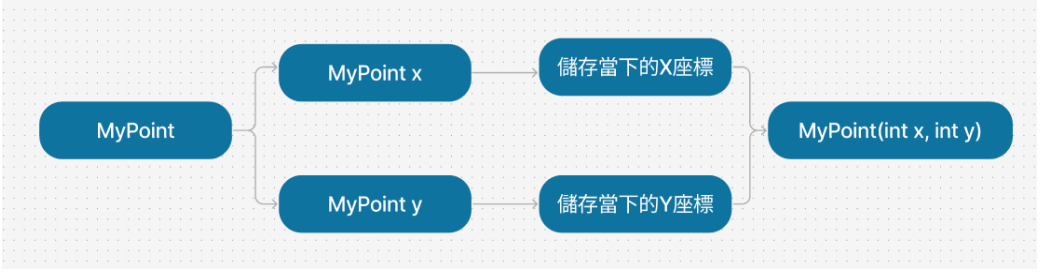
4.支援多顏色棋子變化

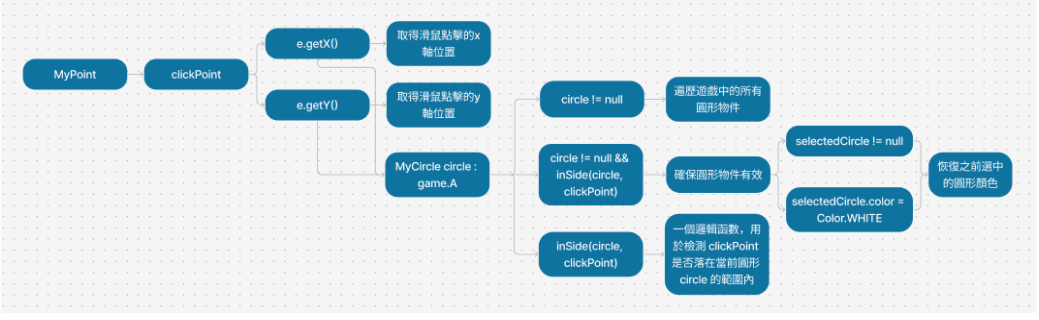
根據遊戲規則分配不同顏色棋子，並在選中或移動過程中更新顏色以提示玩家操作。

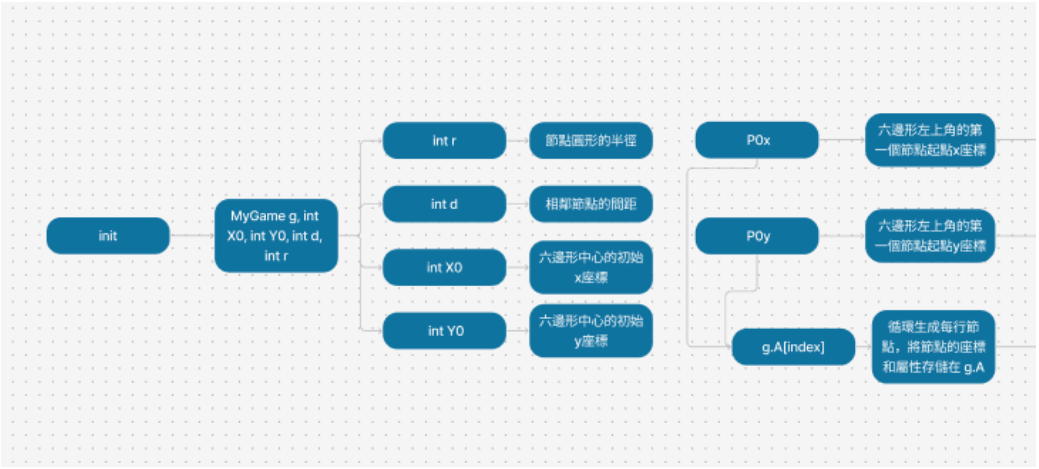
三、系統架構圖


MyGame
int[Cn][8]
靜態常數 Cn 定義節點總數
每個圓最多有 8 個鄰居，代表相鄰圓的索引
MyCircle[] A
存儲所有圓形
int[][] Ac
鄰接陣列，儲存圓形之間的連接資訊
Ac[i][j]
表示圓形 i 是否與圓形 j 連接

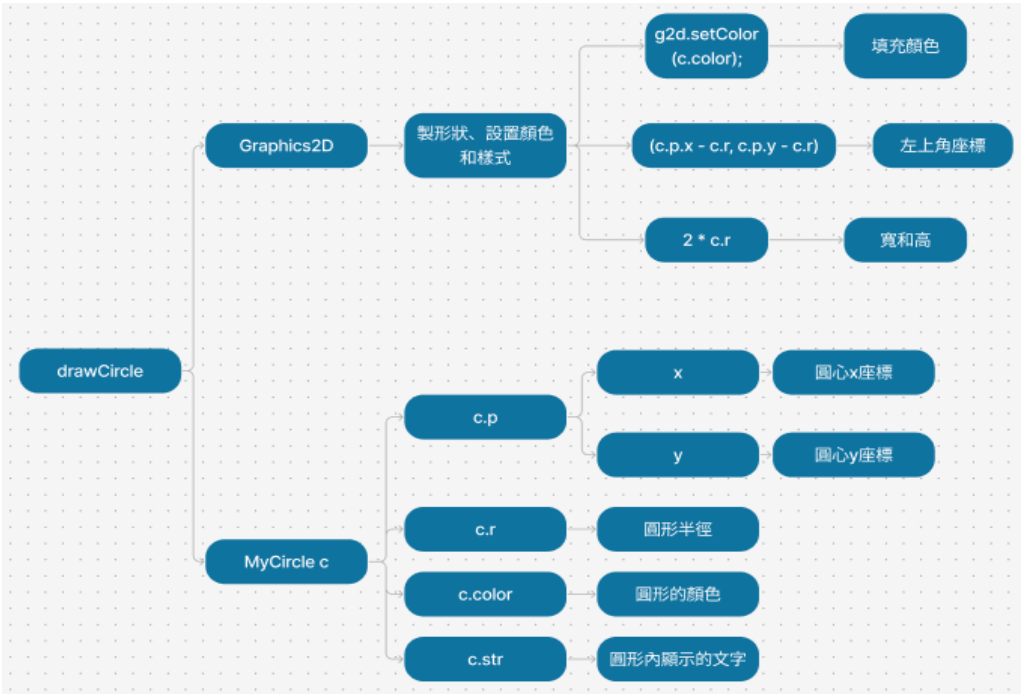

MyCircle
String str
圓的文字標籤
Color color
圓的顏色
MyPoint p
圓心座標
int r
圓的半徑
用於畫圖，每個圓形是幾何結構中的一個節點

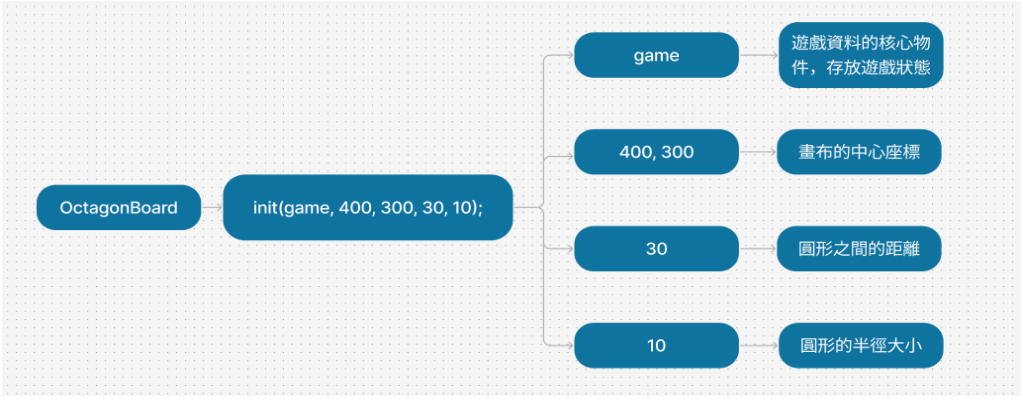

MyPoint
MyPoint x
儲存當下的X座標
MyPoint y
儲存當下的Y座標
MyPoint(int x, int y)

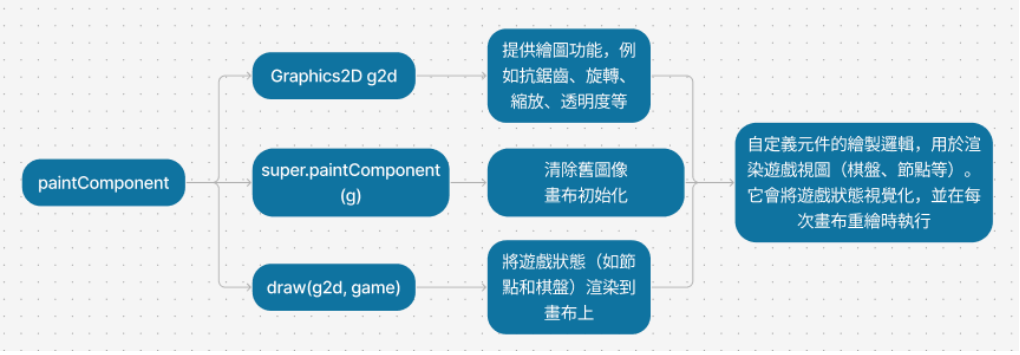

MyPoint
clickPoint
e.getX()
取得滑鼠點擊的x軸位置
e.getY()
取得滑鼠點擊的y軸位置
MyCircle circle : game.A
circle != null
遍歷遊戲中的所有圓形物件
circle != null && inSide(circle, clickPoint)
確保圓形物件有效
selectedCircle != null
selectedCircle.color = Color.WHITE
恢復之前選中的圓形顏色
inSide(circle, clickPoint)
一個邏輯函數，用於檢測 clickPoint 是否落在當前圓形 circle 的範圍內


init
MyGame g, int X0, int Y0, int d, int r
int r
節點圓形的半徑
int d
相鄰節點的間距
int X0
六邊形中心的初始x座標
int Y0
六邊形中心的初始y座標
P0x
六邊形左上角的第一個節點起點x座標
P0y
六邊形左上角的第一個節點起點y座標
g.A[index]
循環生成每行節點，將節點的座標和屬性存儲在 g.A


for (i = 0; i < 9; i++)
i
行索引，從 0 到 8，對應於六邊形網格的每一行
s
與中心行的偏移量，決定當前行的節點縮進
n
當前行的節點數量
x0
當前行的起始節點x座標
y0
當前行的起始節點y座標
int R1
小圓的半徑
int R2
中間圓的外徑
int R
計算每個圓心的半徑
String.valueOf(index)
為節點添加一個標籤，用來識別節點

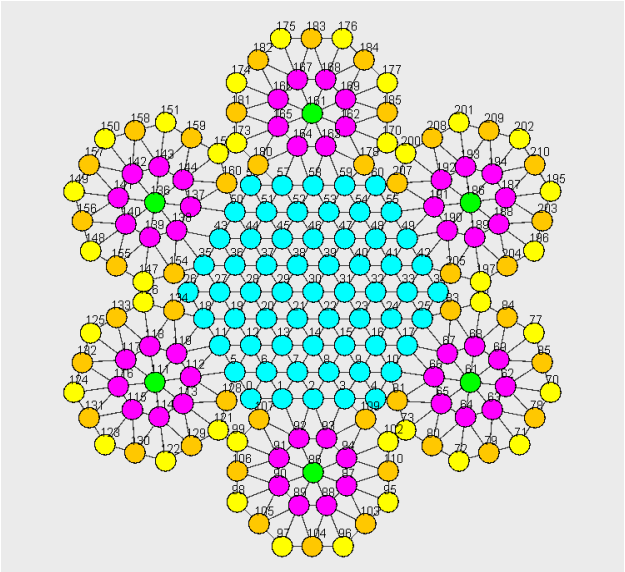

drawCircle
Graphics2D
製形狀、設置顏色和樣式
g2d.setColor(c.color);
填充顏色
(c.p.x - c.r, c.p.y - c.r)
左上角座標
2 * c.r
寬和高
MyCircle c
c.p
x
圓心x座標
y
圓心y座標
c.r
圓形半徑
c.color
圓形的顏色
c.str
圓形內顯示的文字


OctagonBoard
init(game, 400, 300, 30, 10);
game
遊戲資料的核心物件，存放遊戲狀態
400, 300
畫布的中心座標
30
圓形之間的距離
10
圓形的半徑大小



四、棋盤數學公式與邏輯

棋盤的中心由一個六邊形構成，並且外圍環繞著八邊形，根據這一結構，圓形的位置將依據兩個不同的區域來計算：中間六邊形區域和外圍八邊形區域(以下圖顏色區分邏輯)



1.中心六邊形的圓形排列(藍綠色圓點)

- 棋盤共有 9 層
- 每層圓形數量：$n = 9 - |i - 4|$ ，i為當前層索引，$|i - 4|$表示當前層與中心層的偏移量
- 依照左至右，下到上的順序排列
- **X 軸方向的遞增**：每層中的圓形按照水平間距 d排列

  計算公式為：$x_j = x0 \ + j \cdot d$，$j = 0 \sim n - 1$

- **Y 軸方向的遞減**：由下至上排列，座標系統Y 軸從上到下遞增，下到上為遞減

  計算公式為：$y0 = P0y - i{\cdot}d{\cdot}sin(\theta)$，其中 $P0y$ 為整體棋盤的起始 Y 座標(左上)，i 表示當前層索引，θ = 六邊形的內角 = 60度 = $Math.PI / 3$

2.中心六邊形的圓形之間的連接

- **左**$Ac[0]$

  條件: 節點不在當層的左邊界

  $g.Ac[index][0] = index$

- **右**$Ac[1]$

  條件: 節點不在當層的右邊界

  $g.Ac[index][1] = index + 2;$

- **左上**$Ac[2]$

  條件: 節點不在第一層或不在當層的左邊界的下半部節點

  如果節點為下半部的節點 $g.Ac[index][2] = index - n + 1$

  $g.Ac[index][2] = index - n$

- **右上**$Ac[3]$

  條件: 節點不在第一層或不在當層的右邊界的上半部節點

  如果節點為下半部的節點 $g.Ac[index][3] = index - n + 2$

  $g.Ac[index][3] = index - n + 1$

- **左下**$Ac[4]$

  條件: 節點不在最後一層或不在當層的左邊界的上半部節點

  如果節點為下半部的節點 $g.Ac[index][4] = index + n$

  $g.Ac[index][4] = index + n + 1$

- **右下**$Ac[5]$

  條件: 節點不在最後一層或不在當層的右邊界的上半部節點

  如果節點為下半部的節點 $g.Ac[index][5] = index + n + 1$

  $g.Ac[index][5] = index + n + 2$

3.外圍八邊形的圓形排列

- 外圍的八邊形分為三層：中心圓、中間層、外層(外層分為外層圓點與外層邊上中間的圓點)
- **中心圓點(綠色圓點)**

  $x0 = X0 + R{\cdot}cos(\theta{\cdot}i + \theta0)$

  $y0 = Y0 + R{\cdot}sin(\theta{\cdot}i + \theta0)$

  其中R = 整體八邊形的半徑，θ = 360度/八邊形的個數 = 60度 = $Math.PI / 3$，
  $\theta0$ = 偏移角度 = 30度 = $Math.PI / 6$

- **中間層圓點(洋紅色圓點)**

  $x_j = x0 + R1{\cdot}cos(\theta1{\cdot}j + \theta2[i])$

  $y_j = y0 + R1{\cdot}sin(\theta1{\cdot}j + \theta2[i])$

  $R1$ = 中間層的半徑，$\theta1$ = 中間層的角度間隔 = 45度 = $Math.PI / 4$，
  $\theta2[i]$ = 依照偏移角度調整

- **外層圓點(黃色圓點)**

  $x_j = x0 + R2{\cdot}cos(\theta1{\cdot}j + \theta2[i])$

  $y_j = y0 + R2{\cdot}sin(\theta1{\cdot}j + \theta2[i])$

  $R2$ = 外層的半徑，$\theta1$ = 中間層的角度間隔 = 45度 = $Math.PI / 4$，
  $\theta2[i]$ = 依照偏移角度調整

- **外層邊上中間的圓點(橘色圓點)**

  利用中點坐標公式算出中間的圓點座標

4.外圍八邊形的圓形之間的連接

- **中間層跟中心圓連接:** 每個中心圓點與其對應的 8 個中間層圓點連接
- **中間層相鄰的圓連接:** 中間層圓點按順時針方向與左右相鄰節點連接
- **外層圓形與中間層連接:** 外層圓點與對應的中間層圓點連接
- **外層相鄰的圓連接:** 外層圓點按順時針方向與左右相鄰節點連接
- **外層八邊形邊上中間的圓點與中間層八邊形的圓點互相連接:** 外層八邊形邊上中間的圓點與其相鄰的兩個中間層圓點連接

五、跳棋玩法程式分析

玩法互動以點擊事件處理玩家操作是否有效，並根據規則執行對應行為。

玩法規則
1.玩家操作

- 點擊棋子選中後，可移動到周圍相鄰的空白位置
- 棋子僅能移動到周圍相鄰的空白位置(一次一格)

2.遊戲勝利條件

- 當某國王棋子（紅色、綠色或藍色）移動到對應的目標王位時，遊戲結束。
- 王位位置：綠色國王目標位置 - 索引 161；紅色國王目標位置 - 索引 186；藍色國王目標位置 - 索引 136

1.點擊棋子檢查

條件: 判斷滑鼠點擊位置是否在圓形內部

透過圓的標準方程式判斷: $(p_x - c_x)^2 + (p_y - c_y)^2 < r^2$

2.鄰接檢查：設定棋子只能移動到周圍相鄰的位置

條件: 使用迴圈遍歷當前旗子鄰居的矩陣，檢查點擊的目標點是否等於選中旗子的某個鄰居

3.棋子移動邏輯

(1) 玩家點擊一個非空白圓形(點擊棋子)
- 設置該圓形為選中狀態 $isSelected = true$，顏色變為黃色
- 更新選中的棋子 $selectedCircle = circle$，恢復上一個選中圓形顏色及狀態

(2) 玩家點擊一個空白圓形
- 確認存在已選中的棋子 $selectedCircle != null$
- 執行鄰接檢查，判斷是否為合法移動
- 將空白圓形設為選中棋子的顏色
- 將原棋子的位置顏色設置為白色
- 清除選中狀態 $selectedCircle.isSelected = false$
- 清除選中的棋子 $selectedCircle = null$

4.遊戲勝利檢查: 判斷國王棋子是否到達指定王位

條件: 國王顏色等於目標顏色 且 位置索引等於目標索引

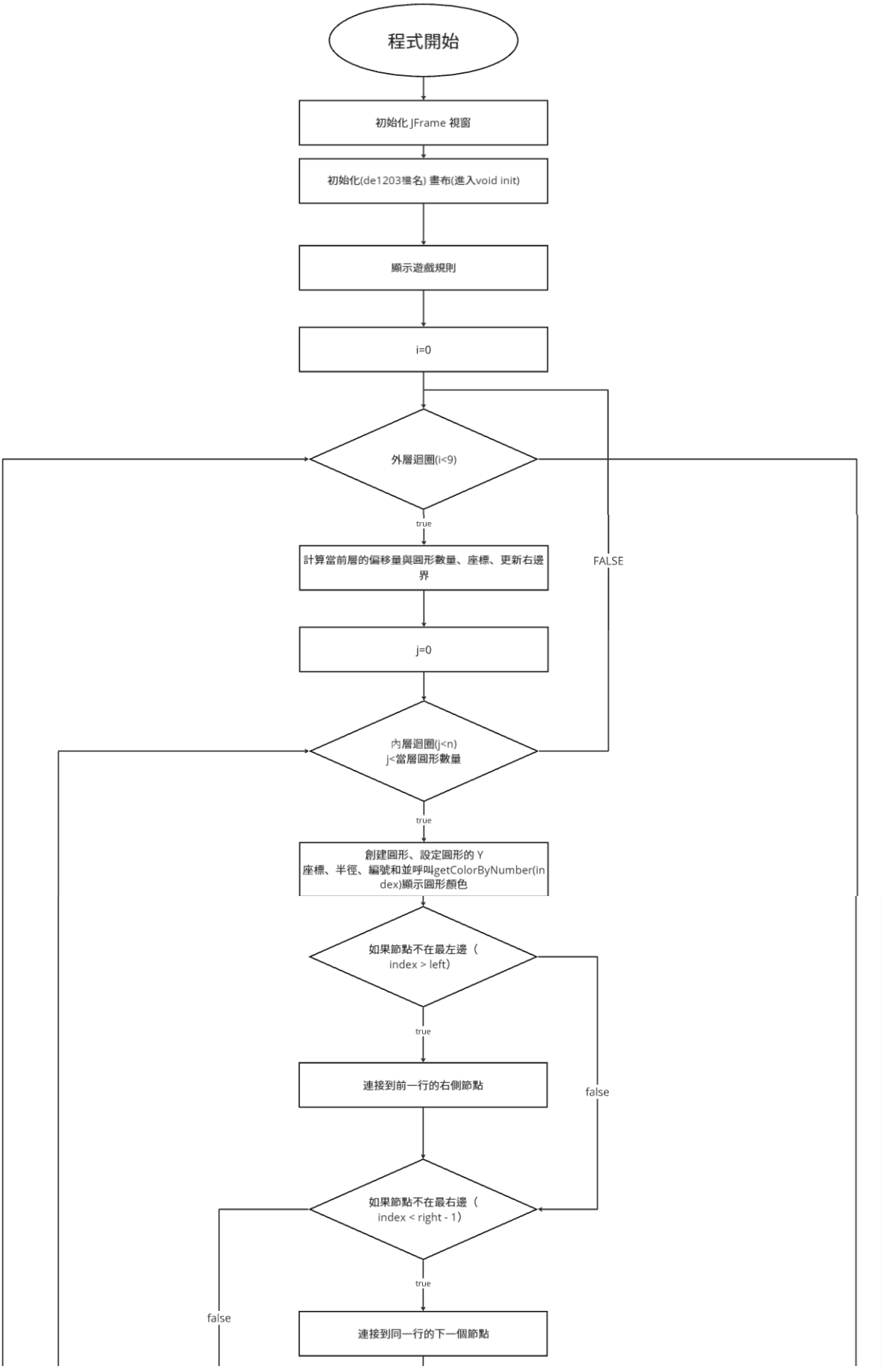
綠色棋子獲勝條件: $circle.color.equals(Color.GREEN)$ && $targetIndex == 161$

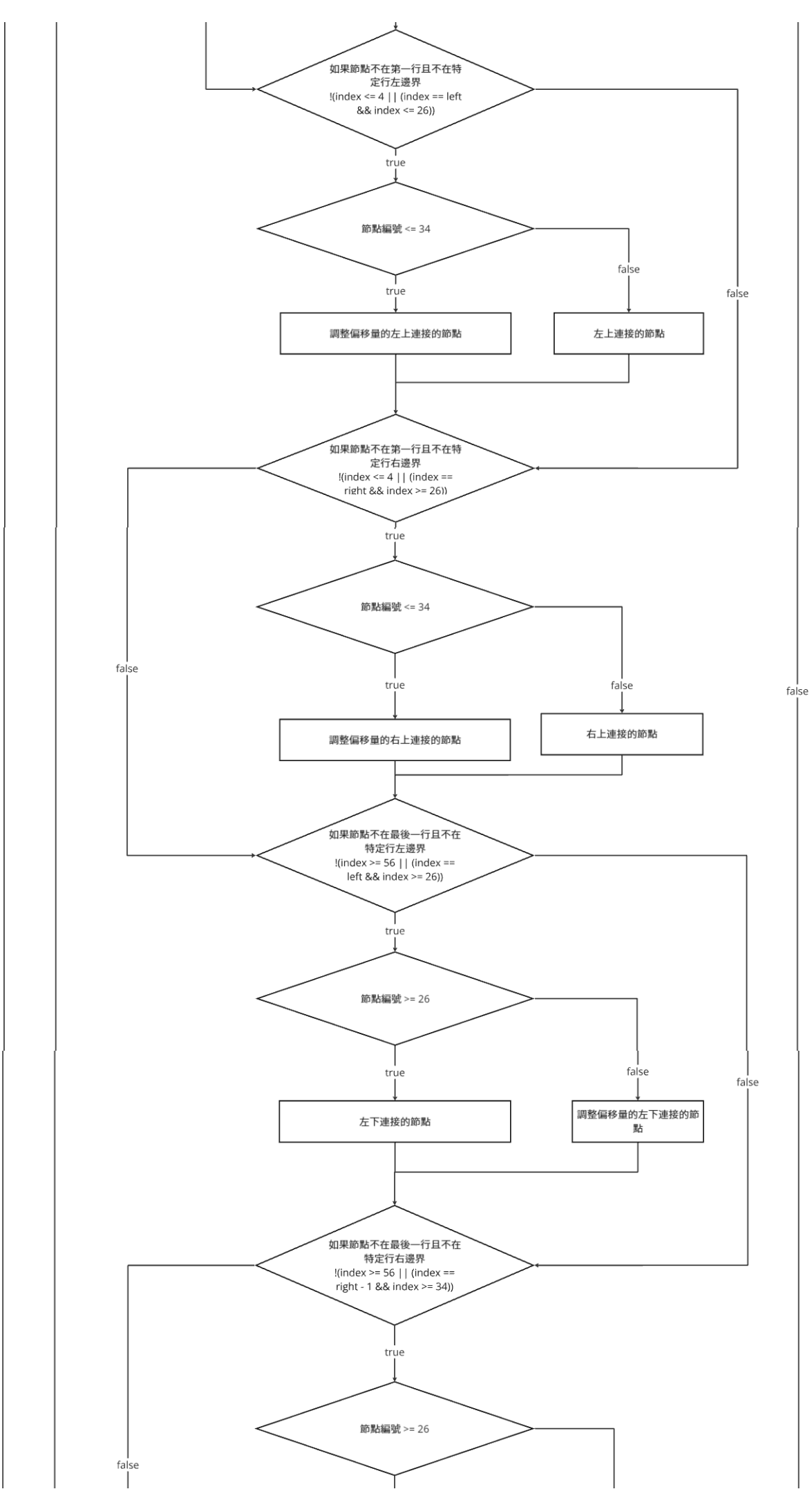
紅色棋子獲勝條件: $circle.color.equals(Color.RED)$ && $targetIndex == 186$

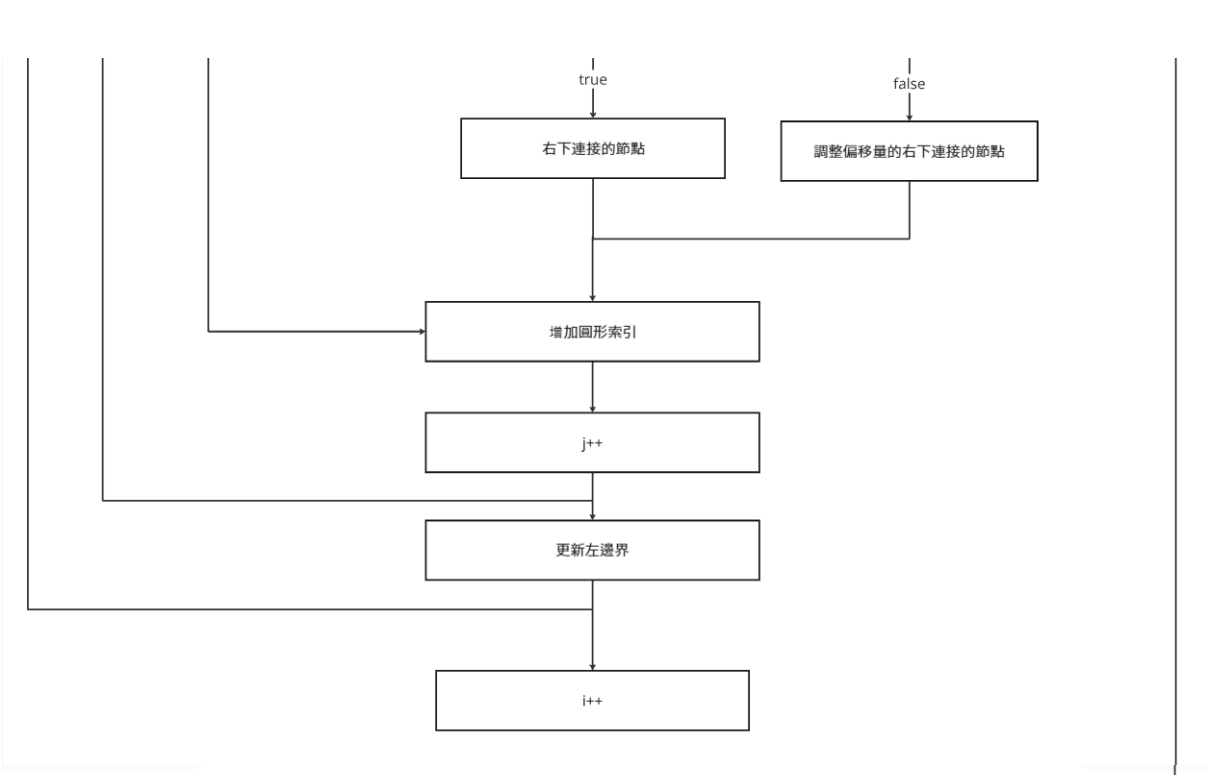
藍色棋子獲勝條件: $circle.color.equals(Color.BLUE)$ && $targetIndex == 136$

當符合條件其中一個顏色獲勝時，彈出提示框，並將 $isGameOver$設為 $true$ ，結束遊戲。

六、系統流程


程式開始
初始化 JFrame 視窗
初始化(de1203檔名) 畫布(進入void init)
顯示遊戲規則
i=0
外層迴圈(i<9)
true
FALSE
計算當前層的偏移量與圖形數量、座標、更新右邊界
j=0
內層迴圈(j<n)
j<當層圖形數量
true
創建圖形、設定圖形的 Y 座標、半徑、編號和並呼叫getColorByNumber(index)顯示圖形顏色
如果節點不在最左邊（index > left）
true
false
連接到前一行的右側節點
如果節點不在最右邊（index < right - 1）
true
false
連接到同一行的下一個節點


如果節點不在第一行且不在特定行左邊界
!(index <= 4 || (index == left && index <= 26))
true
節點編號 <= 34
false
true
調整偏移量的左上連接的節點
左上連接的節點
false
如果節點不在第一行且不在特定行右邊界
!(index <= 4 || (index == right && index >= 26))
true
節點編號 <= 34
false
true
false
調整偏移量的右上連接的節點
右上連接的節點
false
如果節點不在最後一行且不在特定行左邊界
!(index >= 56 || (index == left && index >= 26))
true
節點編號 >= 26
false
true
false
左下連接的節點
調整偏移量的左下連接的節點
false
如果節點不在最後一行且不在特定行右邊界
!(index >= 56 || (index == right - 1 && index >= 34))
true
節點編號 >= 26
false



設定圖形半徑、距離和角度變數

定位八邊形中心圓點
並呼叫getColorByNumber(index)顯示圖形顏色

定位與中心點為基準的中間層八邊形的圓點
並呼叫getColorByNumber(index)顯示圖形顏色

中間層八邊形的圓點與中心圓點連接

中間層八邊形的圓點彼此互相連接

定位外層八邊形的圓點
並呼叫getColorByNumber(index)顯示圖形顏色

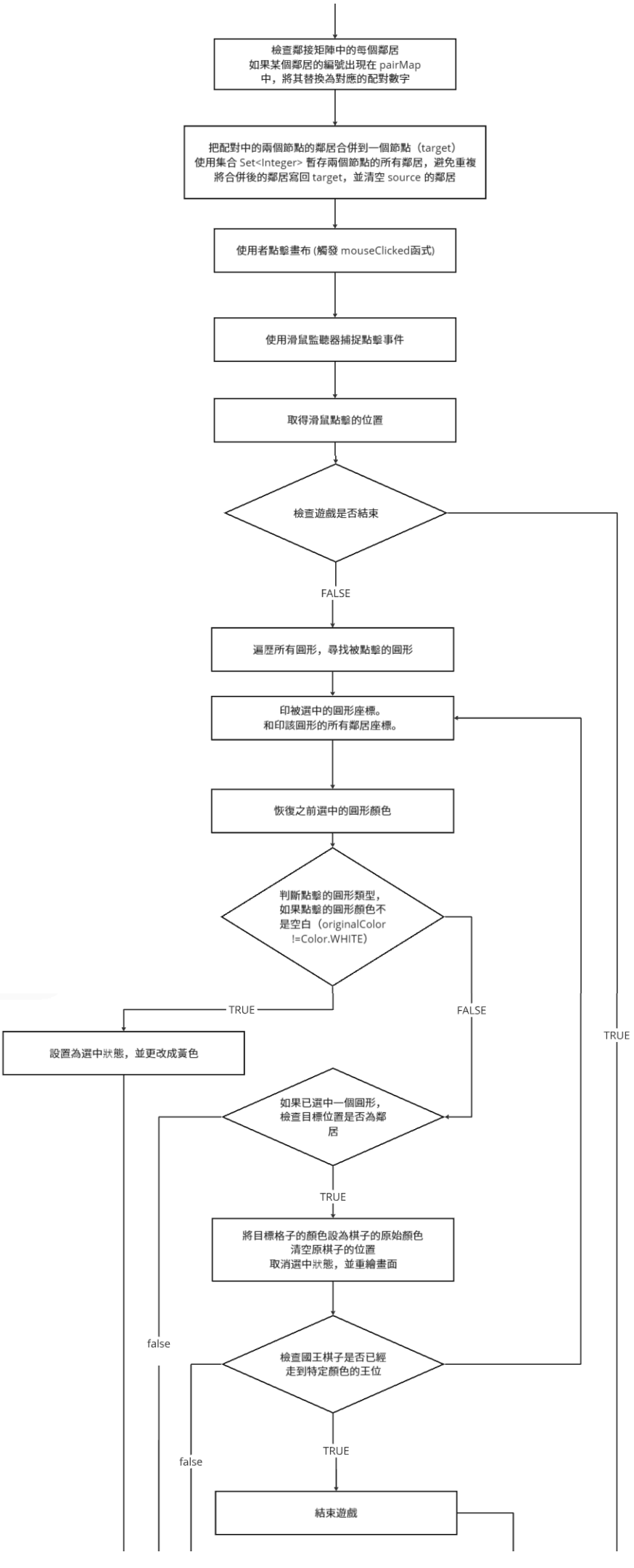
外層八邊形的圓點與中間層八邊形的圓點互相連接

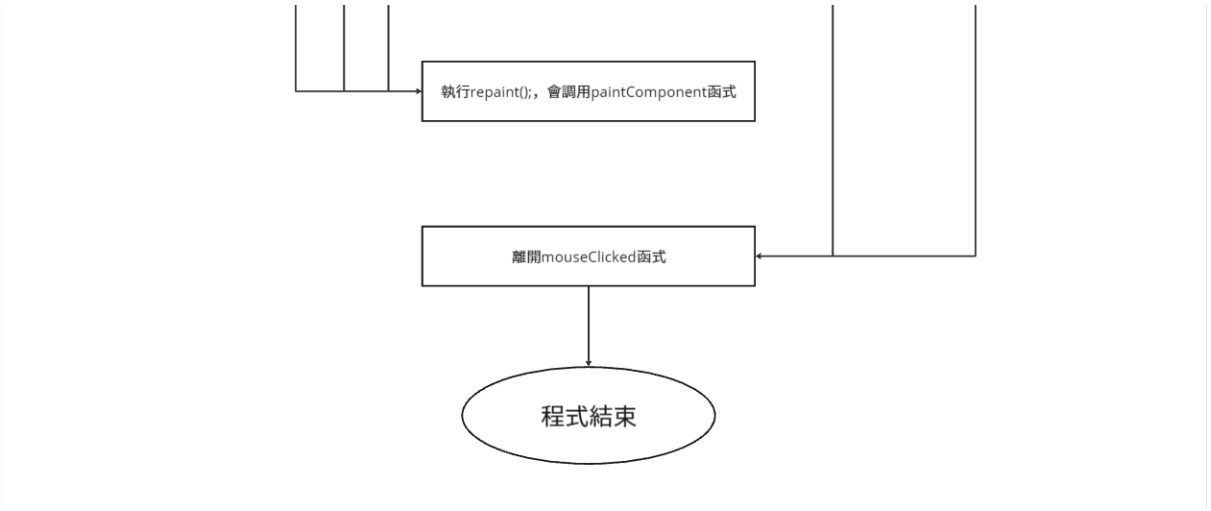
外層八邊形的圓點彼此互相連接

定位外層八邊形邊上中間的圓點
並呼叫getColorByNumber(index)顯示圖形顏色

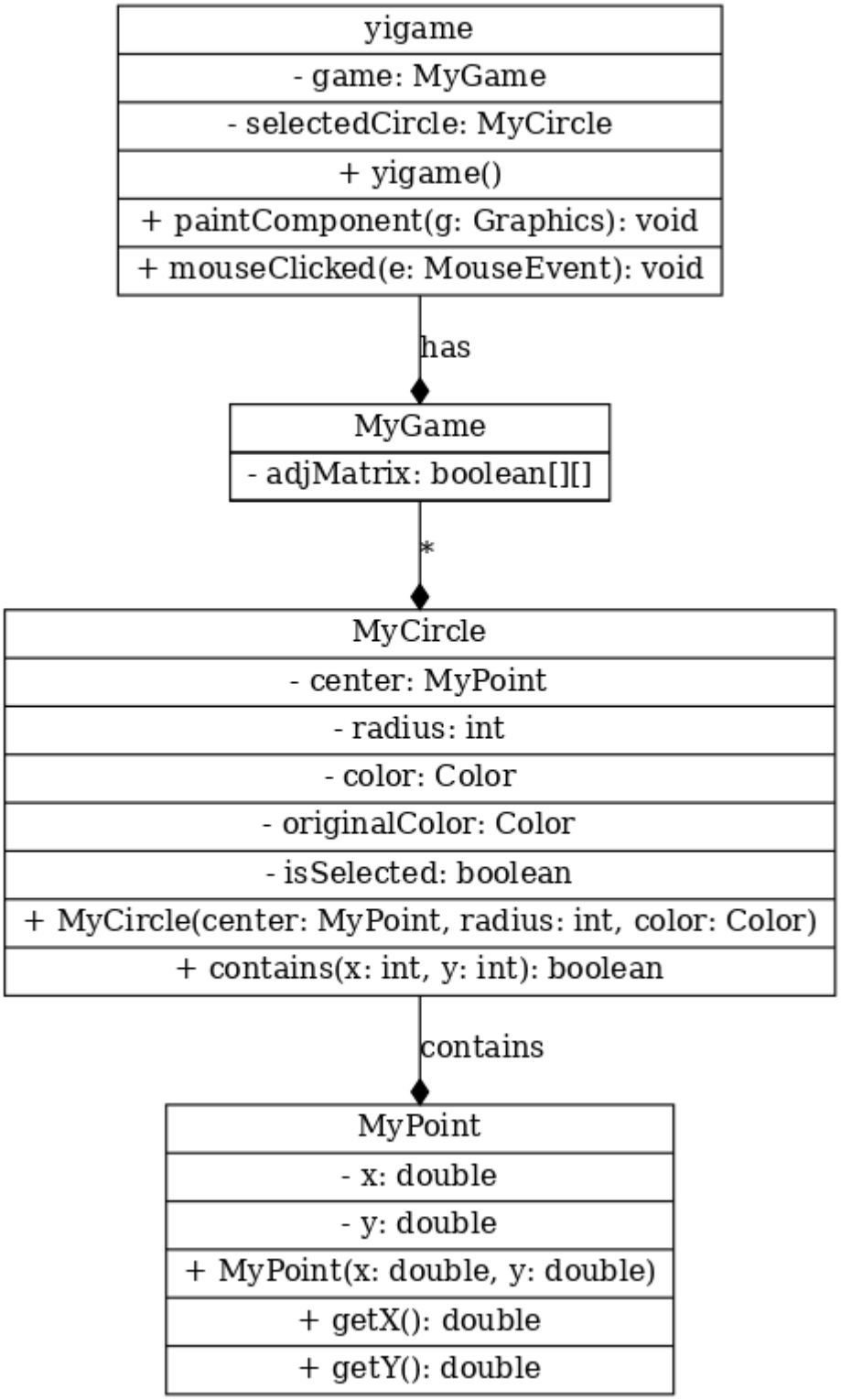
外層八邊形邊上中間的圓點與中間層八邊形的圓點
互相連接

使用 Map<Integer, Integer>
將配對數字中的第一個數字映射到第二個數字


檢查鄰接矩陣中的每個鄰居
如果某個鄰居的編號出現在 pairMap
中，將其替換為對應的配對數字
把配對中的兩個節點的鄰居合併到一個節點（target）
使用集合 Set<Integer> 暫存兩個節點的所有鄰居，避免重複
將合併後的鄰居寫回 target，並清空 source 的鄰居
使用者點擊畫布 (觸發 mouseClicked函式)
使用滑鼠監聽器捕捉點擊事件
取得滑鼠點擊的位置
檢查遊戲是否結束
TRUE
FALSE
遍歷所有圓形，尋找被點擊的圓形
印被選中的圓形座標。
和印該圓形的所有鄰居座標。
恢復之前選中的圓形顏色
判斷點擊的圓形類型，
如果點擊的圓形顏色不
是空白（originalColor
!=Color.WHITE）
TRUE
FALSE
設置為選中狀態，並更改成黃色
如果已選中一個圓形，
檢查目標位置是否為鄰
居
TRUE
將目標格子的顏色設為棋子的原始顏色
清空原棋子的位置
取消選中狀態，並重繪畫面
false
檢查國王棋子是否已經
走到特定顏色的王位
TRUE
false
結束遊戲
TRUE


執行repaint();，會調用paintComponent函式
離開mouseClicked函式
程式結束

七、系統類別圖


yigame
- game: MyGame
- selectedCircle: MyCircle
+ yigame()
+ paintComponent(g: Graphics): void
+ mouseClicked(e: MouseEvent): void
has
MyGame
- adjMatrix: boolean[][]
*
MyCircle
- center: MyPoint
- radius: int
- color: Color
- originalColor: Color
- isSelected: boolean
+ MyCircle(center: MyPoint, radius: int, color: Color)
+ contains(x: int, y: int): boolean
contains
MyPoint
- x: double
- y: double
+ MyPoint(x: double, y: double)
+ getX(): double
+ getY(): double

八、系統程式碼

```
package yigame;

import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Set;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
//定義一個MyPoint 的類別，包含 x 與 y 座標
class MyPoint {
    int x;
    int y;

    public MyPoint(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

//定義一個MyCircle的類別，包含圓心位置、半徑、顏色與顯示的文字
class MyCircle {
    MyPoint p;// 圓心位置
    int r; // 半徑
    Color color;// 顏色
    Color originalColor; // 新增的變量來存儲原本的顏色
    String str;// 圓內顯示的文字
    boolean isSelected; // 新增的布爾變量

    public MyCircle(MyPoint p, int r, Color color, String str) {
        this.p = p;
        this.r = r;
        this.color = color;
        this.str = str;
        this.isSelected = false; // 初始化為未選中
    }
}

//定義 MyGame類別，包含圓的陣列與連線關係的陣列
class MyGame {
```

```
    static final int Cn = 61 + 25 * 6;// 圓的總數量

    MyCircle[] A = new MyCircle[Cn];// 圓的陣列
    int[][] Ac = new int[Cn][8];// 連線的關係矩陣，每個圓最多有 8 個相鄰圓
}

public class yigame extends JPanel {
    MyGame game = new MyGame();
    MyCircle selectedCircle = null; // 記錄當前選中的圓形
    boolean isGameOver = false;
    int[][] pair = { { 74, 10 }, { 75, 25 }, { 82, 17 }, { 100, 1 }, { 101, 3 }, { 108, 2 }, { 120,
5 }, { 127, 18 },
            { 135, 11 }, { 145, 50 }, { 146, 35 }, { 153, 43 }, { 171, 59 }, { 172, 57 }, { 179,
58 }, { 198, 42 },
            { 199, 55 }, { 206, 49 } };// 預設跳過的圓
    // 建構子
    public yigame() {
        init(game, 400, 300, 30, 10);
        addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (isGameOver) {
                    return; // 如果遊戲已經結束，不執行任何操作
                }

                MyPoint clickPoint = new MyPoint(e.getX(), e.getY());
                for (MyCircle circle : game.A) {
                    if (circle != null && inSide(circle, clickPoint)) {
                        // 找到點擊的圓形
                        int currentIndex = findCircleIndex(circle);

                        // 打印當前點及其鄰居
                        System.out.println("Selelected Point " + currentIndex);
                        System.out.print("Neighbors: ");
                        for (int neighbor : game.Ac[currentIndex]) {
                            if (neighbor > 0) { // 確保有鄰居
                                System.out.print((neighbor - 1) + " "); // 打印鄰居的 0-based 索引
                            }
                        }
                        System.out.println();
                        System.out.println("====================================");

                        if (selectedCircle != null) {
                            // 恢復之前選中的圓形顏色
```

```
                    selectedCircle.isSelected = false;
                    selectedCircle.color = selectedCircle.originalColor;
                }

                // 如果點擊的是一個有效的旗子
                if (circle.originalColor != Color.WHITE) {
                    // 設定新選中的圓形顏色為黃色
                    circle.isSelected = true;
                    circle.color = Color.YELLOW;
                    selectedCircle = circle; // 更新選中的圓形
                    repaint();// 更新畫面
                } else if (selectedCircle != null) {
                    // 確保目標點是相鄰點
                    currentIndex = findCircleIndex(selectedCircle);// 找到選中圓形的索
引
                    int targetIndex = findCircleIndex(circle);// 找到目標圓形的索引
                    if (isAdjacent(currentIndex, targetIndex)) {
                        // 點擊空白位置，移動旗子
                        circle.color = selectedCircle.originalColor; // 將顏色設為選中的旗
子的顏色
                        circle.originalColor = selectedCircle.originalColor;

                        // 將之前的旗子位置設為白色
                        selectedCircle.color = Color.WHITE;
                        selectedCircle.originalColor = Color.WHITE;
                        selectedCircle.isSelected = false;
                        selectedCircle = null; // 清除選中的旗子
                        repaint();// 更新畫面

                        // 檢查國王棋子是否已經走到特定顏色的王位
                        if ((circle.color.equals(Color.GREEN) && targetIndex == 161) ||
                            (circle.color.equals(Color.RED) && targetIndex == 186) ||
                            (circle.color.equals(Color.BLUE) && targetIndex == 136)) {
                            String colorName = circle.color.equals(Color.GREEN) ? "綠色"
:
                                               circle.color.equals(Color.RED) ? "紅色" :
                                               "藍色";
                            JOptionPane.showMessageDialog(null, colorName + "國王棋
子已經走到王位！");
                            isGameOver = true; // 設置遊戲結束狀態
                            JOptionPane.showMessageDialog(null, "遊戲結束！請關閉視
窗重新開始遊戲。");
                        }
                    }
```

```
                } else {
                    repaint();// 更新畫面
                }
                break;
            }
        }
    }

    private int findCircleIndex(MyCircle circle) {
        for (int i = 0; i < game.A.length; i++) {
            if (game.A[i] == circle) {
                return i;
            }
        }
        return -1; // 未找到
    }

    private boolean isAdjacent(int currentIndex, int targetIndex) {
        if (currentIndex == -1 || targetIndex == -1) {
            return false;
        }
        for (int neighbor : game.Ac[currentIndex]) {
            if (neighbor - 1 == targetIndex) { // 鄰接矩陣是 1-based 索引
                return true;
            }
        }
        return false;
    }
});
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);//調用父類的方法以清除畫布，確保新的內容不會疊加到舊的畫面上。
    Graphics2D g2d = (Graphics2D) g;//將 Graphics 轉型為 Graphics2D，以便使用更高級的繪圖功能。
    Set<String> drawnLines = new HashSet<>();//用來儲存已繪製的線段，避免重複繪製。
    int k = 0;//初始化變數 k，用於遍歷 pair 陣列。

    //繪製所有的連接線
    for (int i = 0; i < MyGame.Cn; i++) {
        if (k < pair.length && pair[k][0] == i) {
```

```
                k++;
                continue;
            }
            for (int j = 0; j < 8; j++) {
                if (game.Ac[i][j] > 0) {
                    String lineKey = i + "-" + game.Ac[i][j];
                    String reverseLineKey = game.Ac[i][j] + "-" + i;
                    if (!drawnLines.contains(lineKey) &&
!drawnLines.contains(reverseLineKey)) {
                        g2d.drawLine(game.A[i].p.x, game.A[i].p.y, game.A[game.Ac[i][j] -
1].p.x, game.A[game.Ac[i][j] - 1].p.y);
                        drawnLines.add(lineKey);
                    }
                }
            }
        }

        //跳過特定節點
        int[] skipIndices = { 74, 75, 82, 100, 101, 108, 120, 127, 135, 145, 146, 153,
171, 172, 179, 198, 199, 206 };
        Set<Integer> skipSet = new HashSet<>();
        for (int index : skipIndices) {
            skipSet.add(index);
        }
        //繪製圓形節點
        for (int i = 0; i < MyGame.Cn; i++) {
            if (game.A[i] != null && !skipSet.contains(i)) {
                drawCircle(g2d, game.A[i]);
            }
        }
    }
    // 繪製圓形
    void drawCircle(Graphics2D g2d, MyCircle c) {
        g2d.setColor(c.color);
        g2d.fill(new Ellipse2D.Double(c.p.x - c.r, c.p.y - c.r, 2 * c.r, 2 * c.r));// 填充圓形

        g2d.setColor(Color.BLACK);// 設置邊框顏色
        g2d.draw(new Ellipse2D.Double(c.p.x - c.r, c.p.y - c.r, 2 * c.r, 2 * c.r));// 繪製圓形
邊框

        g2d.drawString(c.str, c.p.x - c.r + 5, c.p.y - c.r + 2);// 在圓內顯示文字

    }
```

```
    private Color getColorByNumber(int number) {
        int[] greenNumbers = { 1, 2, 3, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
100, 101, 102, 103, 104, 105, 106, 107,
                108, 109, 110 };// 綠色普通棋子
        int[] greenKingNumbers = { 86 };//綠色國王棋子

        int[] blueNumbers = { 10, 17, 25, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
74, 75, 76, 77, 78, 79, 80, 81, 82,
                83, 84, 85 };// 藍色普通棋子
        int[] blueKingNumbers = { 61 };//藍色國王棋子

        int[] redNumbers = { 5, 11, 18, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121,
122, 123, 124, 125, 126, 127, 128, 129,
                130, 131, 132, 133, 134, 135 };// 紅色普通棋子
        int[] redKingNumbers = { 111 };//紅色國王棋子


        for (int num : greenNumbers) {// 遍歷綠色普通棋子數字的陣列
            if (number == num) {// 如果當前數字與綠色普通棋子數字相符
                return new Color(50, 200, 100);//就顯示RGB(50,200,100)顏色
            }
        }


        for (int num : greenKingNumbers) {// 遍歷綠色國王數字的陣列
            if (number == num) {// 如果當前數字與綠色國王數字相符
                return Color.GREEN;//就顯示綠色
            }
        }

        for (int num : blueNumbers) {// 遍歷藍色普通棋子數字的陣列
            if (number == num) {// 如果當前數字與藍色普通棋子數字相符
                return new Color(50, 100, 200);//就顯示RGB(50,100,200)顏色
            }
        }

        for (int num : blueKingNumbers) {// 遍歷藍色國王數字的陣列
            if (number == num) {// 如果當前數字與藍色國王數字相符
                return Color.BLUE;//就顯示藍色
            }
        }

        for (int num : redNumbers) {// 遍歷紅色普通棋子數字的陣列
            if (number == num) {// 如果當前數字與紅色普通棋子數字相符
```

```
            return new Color(200, 50, 100);//就顯示RGB(200,50,100)顏色
        }
    }

    for (int num : redKingNumbers) { // 遍歷紅色國王數字的陣列
        if (number == num) {// 如果當前數字與紅色國王數字相符
            return Color.RED;//就顯示紅色
        }
    }

    return Color.WHITE;// 預設顏色（空白）
}
// 判斷點是否在圓內
boolean inSide(MyCircle c, MyPoint p) {
    return ((c.p.x - p.x) * (c.p.x - p.x) + (c.p.y - p.y) * (c.p.y - p.y)) < c.r * c.r;
}



public void init(MyGame g, int X0, int Y0, int d, int r) {
    ////// 六角形
    double theta = Math.PI / 3;// 六邊形角度
 // 設定圓形結構的起始位置 P0x, P0y
    int P0x = X0 - 4 * d;
    int P0y = Y0 + (int) (4 * d * Math.sin(theta));

 // 宣告各種變數，用來控制圓形生成的過程
    int i, s, n, j, x0, y0, left = 0, right = 0, index = 0;

 // 生成中間六邊形
    for (i = 0; i < 9; i++) {// i 迴圈用來生成 9 層圓形

        s = Math.abs(i - 4);// 計算當前層與中心的偏移量

        n = 9 - s;// 當前層的圓形數量

        x0 = P0x + s * d / 2;// 計算每個圓形的 X 座標

        y0 = (int) (P0y - i * d * Math.sin(theta));// 計算每個圓形的 Y 座標

        right += n;// 更新右邊界
        for (j = 0; j < n; j++) {// j 迴圈用來生成當前層的圓形
            g.A[index] = new MyCircle(new MyPoint(x0 + d * j, y0), r,
getColorByNumber(index), String.valueOf(index));// 根據 index 獲得顏色
```

```
            g.A[index].originalColor = g.A[index].color;

            // 設定相鄰圓形的關聯
            if (index > left)
                g.Ac[index][0] = index;
            if (index < right - 1)
                g.Ac[index][1] = index + 2;
            if (!(index <= 4 || (index == left && index <= 26))) {
                if (index <= 34)
                    g.Ac[index][2] = index - n + 1;
                else
                    g.Ac[index][2] = index - n;
            }
            if (!(index <= 4 || (index == right && index >= 26))) {
                if (index <= 34)
                    g.Ac[index][3] = index - n + 2;
                else
                    g.Ac[index][3] = index - n + 1;
            }
            if (!(index >= 56 || (index == left && index >= 26))) {
                if (index >= 26)
                    g.Ac[index][4] = index + n;
                else
                    g.Ac[index][4] = index + n + 1;
            }
            if (!(index >= 56 || (index == right - 1 && index >= 34))) {
                if (index >= 26)
                    g.Ac[index][5] = index + n + 1;
                else
                    g.Ac[index][5] = index + n + 2;
            }
            index++; // 增加圓形索引
            System.out.println("Point " + (index - 1) + " coordinates: (" + g.A[index -
1].p.x + ", "
                    + g.A[index - 1].p.y + ")");
        }
        System.out.println("====================================");
        left += n;// 更新左邊界
    }

    ////// 六個八邊形的圓形
    int R1 = d + 5;
    // 計算第一層圓點與中心點的距離 R1，基於給定距離 d 並增加 5 作為半徑。
```

```
        int R2 = (int) (d / Math.sin(Math.PI / 8));
        // 計算第二層圓點的距離 R2，根據圓的幾何公式：d / sin(π/8)。
        int R = (int) (R2 * Math.cos(Math.PI / 8) + 4 * d * Math.sin(theta));
        // 計算主圓形半徑 R，將 R2 的部分餘弦值與其他調整量結合。
        theta = Math.PI / 3;     // 設定主迴圈角度增量 θ 為 π/3 (60°)。
        double theta0 = Math.PI / 6;
        // 初始偏移角度 θ₀ 設為 π/6 (30°)。

        double[] theta2 = { Math.PI / 24, Math.PI / 8, -Math.PI / 24, Math.PI / 24, Math.PI
/ 8, -Math.PI / 24 };
    // 定義每次主迴圈使用的角度調整陣列，用於讓每一圈的圓點具有不同的排列角
度。

        for (i = 0; i < 6; i++) {// 主迴圈執行 6 次，對應六邊形的 6 個頂點。

            x0 = (int) (X0 + R * Math.cos(theta * i + theta0));
            // 計算每個頂點的 x 座標，基於圓心 (X0, Y0)，半徑 R 和角度偏移。

            y0 = (int) (Y0 + R * Math.sin(theta * i + theta0));
         // 計算每個頂點的 y 座標。

            g.A[index] = new MyCircle(new MyPoint(x0, y0), r, getColorByNumber(index),
String.valueOf(index));//根據 index 指定該圓形的顏色。
            g.A[index].originalColor = g.A[index].color;

            int center = index;// 記錄當前頂點的索引，作為中心點。

            index++;// 更新索引，準備下一個頂點。

            double theta1 = Math.PI / 4;// 定義第一層圓點的角度間距，每 45 度 (π/4) 一
個點。
            for (j = 0; j < 8; j++) {// 第一層迴圈，圍繞中心點建立 8 個圓點。

                g.A[index] = new MyCircle(new MyPoint((int) (x0 + R1 * Math.cos(theta1 * j
+ theta2[i])),
                        (int) (y0 + R1 * Math.sin(theta1 * j + theta2[i]))),
                        r, getColorByNumber(index), String.valueOf(index));// 計算並新增每個
圓點的座標與顏色。
                        g.A[index].originalColor = g.A[index].color;
                index++;// 更新索引，指向下一個圓點。

                System.out.println("Point " + (index - 1) + " coordinates: (" + g.A[index -
1].p.x + ", "
                        + g.A[index - 1].p.y + ")");// 輸出當前圓點的座標資訊。
```

```
            g.Ac[center][j] = index;// 設定中心點與圓點的連接關係。

            g.Ac[index - 1][0] = center + 1;// 建立雙向連接：圓點連回中心點。

        }
        System.out.println("====================================");
        for (j = 0; j < 8; j++) {// 設定第一層圓點之間的連接關係。

            g.Ac[center + 1 + j][1] = center + 2 + (j + 1) % 8;// 圓點的下一個點。

            g.Ac[center + 1 + j][2] = center + 2 + (j + 7) % 8;// 圓點的上一個點。

            g.Ac[center + 1 + j][3] = center + j + 8 + 1 + 1;
            g.Ac[center + 1 + j][4] = center + j + 15 + 2;

         // 圓點的其他連接點（依邏輯延續設置）。
            if (j == 0)
                g.Ac[center + 1 + j][5] = center + j + 24 + 1;// 特殊處理第一個點的附加連
接。
            else
                g.Ac[center + 1 + j][5] = center + j + 16 + 1;// 其他圓點的附加連接。
        }
        int start = index;// 記錄第二層圓點起始索引。

        for (j = 0; j < 8; j++) {// 第二層迴圈，建立外圈圓點。

            g.A[index] = new MyCircle(new MyPoint((int) (x0 + R2 * Math.cos(theta1 * j
+ theta2[i])),
                    (int) (y0 + R2 * Math.sin(theta1 * j + theta2[i]))),
                    r, getColorByNumber(index), String.valueOf(index));
                    g.A[index].originalColor = g.A[index].color;
            g.Ac[index][0] = index - 8 + 1;// 設定第二層圓點與第一層圓點的連接。

            index++;// 更新索引。
            System.out.println("Point " + (index - 1) + " coordinates: (" + g.A[index -
1].p.x + ", "
                    + g.A[index - 1].p.y + ")");
        }
        System.out.println("====================================");
        for (j = 0; j < 8; j++) {//設定第二層圓點之間的連接。

            g.Ac[start + j][1] = start + j + 7 + 1;// 下一個點。
```

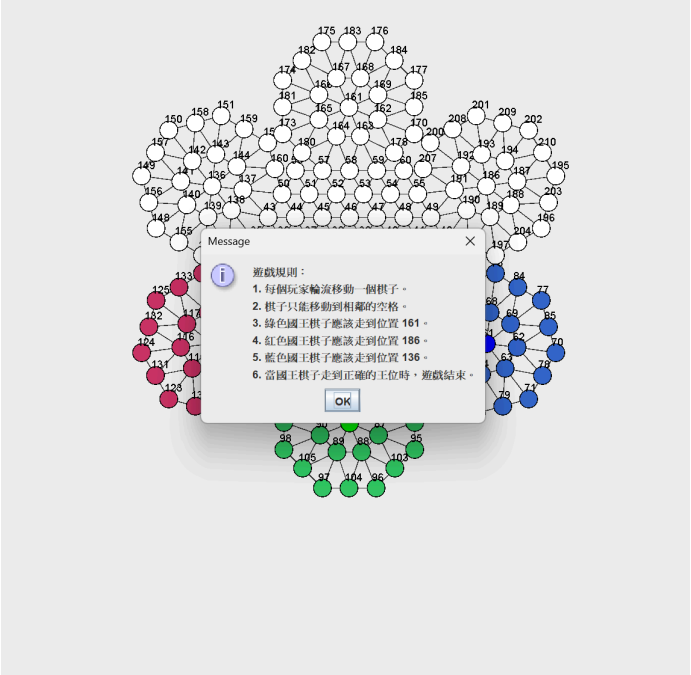
```
            g.Ac[start + j][2] = start + j + 8 + 1;// 上一個點。

        }
        for (j = 0; j < 8; j++) {// 計算第二層圓點之間的中點，並新增圓點。

            g.A[index] = new MyCircle(new MyPoint(
                    (g.A[start + j].p.x + g.A[start + (j + 1) % 8].p.x) / 2,
                    (g.A[start + j].p.y + g.A[start + (j + 1) % 8].p.y) / 2),
                    r, getColorByNumber(index), String.valueOf(index));
                    g.A[index].originalColor = g.A[index].color;
            if (j < 7) {// 處理前 7 個圓點中點的連接。

                g.Ac[index][0] = index - 15 + 1;
                g.Ac[index][1] = index - 16 + 1;
                g.Ac[index - 15][4] = index + 1;
                g.Ac[index - 16][4] = index + 1;
                g.Ac[index][2] = index - 7 + 1;
                g.Ac[index][3] = index - 8 + 1;
                g.Ac[index - 7][5] = index + 1;
                g.Ac[index - 8][5] = index + 1;
            } else {// 處理最後一個圓點的中點。

                g.Ac[index][0] = index - 23 + 1;
                g.Ac[index][1] = index - 16 + 1;
                g.Ac[index - 23][4] = index + 1;
                g.Ac[index - 16][4] = index + 1;
                g.Ac[index][2] = index - 15 + 1;
                g.Ac[index][3] = index - 8 + 1;
                g.Ac[index - 15][5] = index + 1;
                g.Ac[index - 8][5] = index + 1;
            }
            index++;
            System.out.println("Point " + (index - 1) + " coordinates: (" + g.A[index -
1].p.x + ", "
                    + g.A[index - 1].p.y + ")");
        }
        System.out.println("====================================");
    }

    // 建立配對的映射表（僅將第一個數字映射到第二個數字）
    Map<Integer, Integer> pairMap = new HashMap<>();
    for (int[] p : pair) {
        pairMap.put(p[0], p[1]); // 僅記錄第一格到第二格的映射
    }
```

```
    // 遍歷所有點的鄰居
    for (i = 0; i < g.Ac.length; i++) {
        for (j = 0; j < 8; j++) {// 檢查節點的 8 個鄰居
            if (g.Ac[i][j] > 0) { // 確保鄰居有效
                int neighbor = g.Ac[i][j] - 1; // 1-based 索引轉換為 0-based
                if (pairMap.containsKey(neighbor)) {
                    // 如果鄰居是配對的第一個數字，替換成第二個數字
                    g.Ac[i][j] = pairMap.get(neighbor) + 1; // 轉換回 1-based 索引
                }
            }
        }
    }

    // 合併鄰居
    for (int[] p : pair) {
        int source = p[0]; // 第一個數字
        int target = p[1]; // 第二個數字

        // 合併 `source` 的鄰居到 `target`
        Set<Integer> neighbors = new HashSet<>();
        for (j = 0; j < 8; j++) {
            if (g.Ac[target][j] > 0) {
                neighbors.add(g.Ac[target][j] - 1); // 1-based 索引轉換為 0-based
            }
            if (g.Ac[source][j] > 0) {
                neighbors.add(g.Ac[source][j] - 1);
            }
        }

        // 將合併後的鄰居重新寫回 `target`
        int k = 0;
        for (int neighbor : neighbors) {
            g.Ac[target][k++] = neighbor + 1; // 轉換回 1-based 索引
            if (k >= 8)
                break; // 最多支持 8 個鄰居
        }

        // 清空 `source` 的鄰居
        for (j = 0; j < 8; j++) {
            g.Ac[source][j] = 0;
        }
    }
}
```

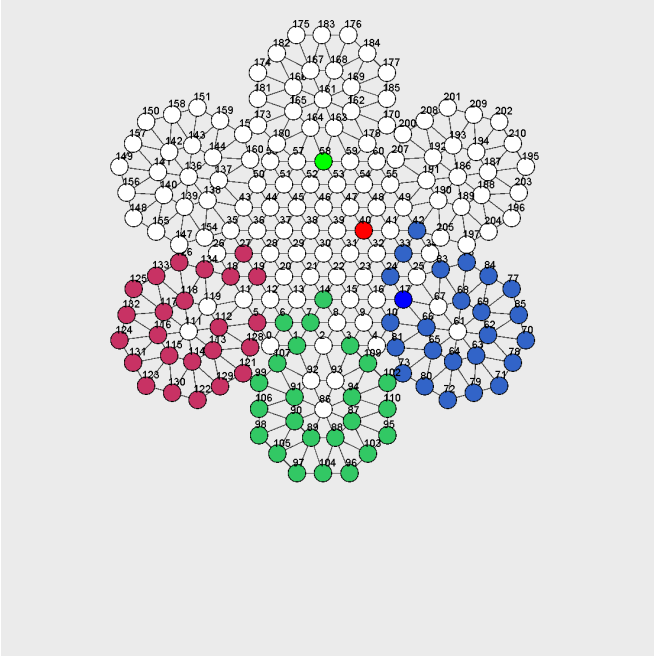
```
    // 繪製所有內容
    void draw(Graphics2D g2d, MyGame g) {
        Set<String> drawnLines = new HashSet<>();
        int k = 0;
        for (int i = 0; i < MyGame.Cn; i++) {
            if (k < pair.length && pair[k][0] == i) {
                k++;
                continue;
            }
           // 繪製每條線，避免重複
            for (int j = 0; j < 8; j++) {
                if (g.Ac[i][j] > 0) {
                    String lineKey = i + "-" + g.Ac[i][j];
                    String reverseLineKey = g.Ac[i][j] + "-" + i;
                    if (!drawnLines.contains(lineKey) &&
!drawnLines.contains(reverseLineKey)) {
                        g2d.drawLine(g.A[i].p.x, g.A[i].p.y, g.A[g.Ac[i][j] - 1].p.x, g.A[g.Ac[i][j] -
1].p.y);
                        drawnLines.add(lineKey);
                    }
                }
            }
        }
    }

    // 主程式進入點
    public static void main(String[] args) {
        JFrame frame = new JFrame("yigame");// 建立主視窗
        yigame board = new yigame();
        frame.add(board);
        frame.setSize(800, 800);// 設置視窗大小
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null); // 將視窗設置在螢幕中央
        frame.setVisible(true);
        // 顯示遊戲規則
        String gameRules = "遊戲規則：\n"
                + "1. 每個玩家輪流移動一個棋子。\n"
                + "2. 棋子只能移動到相鄰的空格。\n"
                + "3. 綠色國王棋子應該走到位置 161。\n"
                + "4. 紅色國王棋子應該走到位置 186。\n"
                + "5. 藍色國王棋子應該走到位置 136。\n"
                + "6. 當國王棋子走到正確的王位時，遊戲結束。";
        JOptionPane.showMessageDialog(frame, gameRules);
```

```
    }
}
```

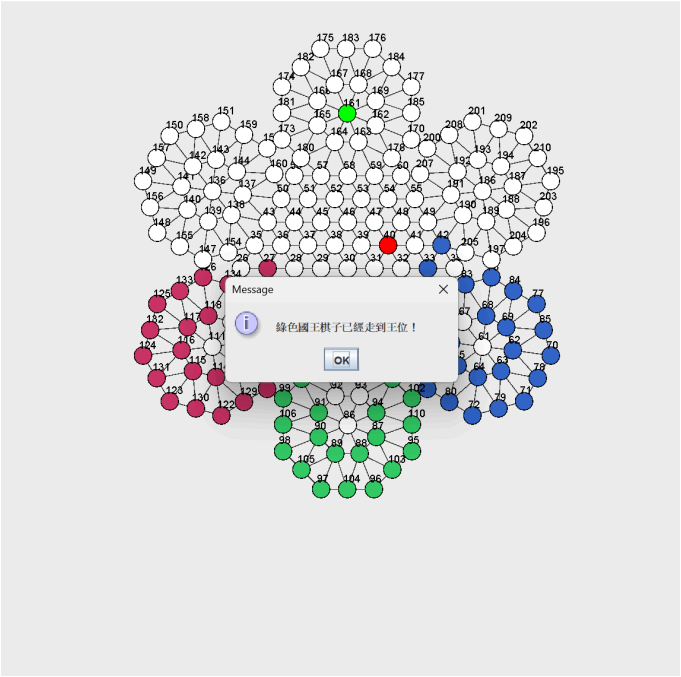
九、程式執行圖



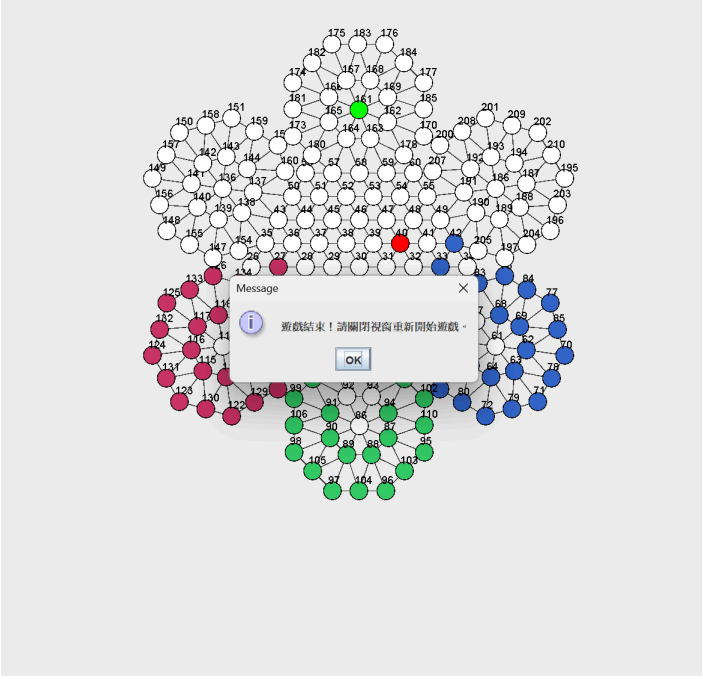
遊戲介紹



玩家遊玩中



其中一個玩家的國王旗子走到王位



遊戲結束