Högskolan i Gävle

Labyrint Projekt

John Engblom Sandin
engblomsandinjohn@gmail.com
19joen03

William Tiderman
williamtiderman@live.se
19witi01

2021-01-11

Kurs: Algoritmer och datastrukturer 7.5hp

Lärare: Anders Jackson
Lärare/handledare: Hanna Holmgren

Innehåll

Inledning

Projektet är att skapa labyrinter och de omfattar generering och lösning av olika typer av labyrinter. De ska visa hur algoritmer bygger och löser labyrinter, samt visa hur effektiv de olika typerna är beroende på hur bra typens algoritm är skapad.

MazeGeneration

För att kunna använda sig av olika typer av algoritmer så implementeras ett interface kallat **MazeGeneration**. Detta interface deklarerar de metoder som en lösningsalgoritm kommer behöva för att lösa en labyrint. Detta projekt använder sig av tre olika algoritmer för att instansiera labyrinter: En rekursiv backtracker, en rekursiv sidewinder och ett sätt för användaren att läsa in egna labyrinter. Alla tre använder sig av fyra 2-dimensionella boolean listor som håller koll på om det finns en vägg i varje riktning för varje koordinat. Men först kan vi prata om den rekursiva backtrackern.

MazeGenRecBack

Den första algoritmen för labyrint generation är en rekursiv backtracker som slumpmässigt väljer en plats sedan väljer varje steg slumpmässigt tills alla koordinater har blivit ritade. Detta skapar en helt slumpad labyrint som inte har någon vertikal eller horisontell påverkning.
Vid instansieringen av en labyrint instansieras även alla listor dessutom tilldelas alla celler väggar i alla fyra riktningar, denna metod ses i figur 1.

```
private void init() {
    north = new ArrayList<List<Boolean>>();
    east = new ArrayList<List<Boolean>>();
    west = new ArrayList<List<Boolean>>();
    south = new ArrayList<List<Boolean>>();
    visited = new ArrayList<List<Boolean>>();

    for(int i = 0; i < n+2; i++){
        north.add(new ArrayList<Boolean>());
        east.add(new ArrayList<Boolean>());
        west.add(new ArrayList<Boolean>());
        south.add(new ArrayList<Boolean>());
        visited.add(new ArrayList<Boolean>());

        for(int j = 0; j < n+2; j++) {
            north.get(i).add(true);
            east.get(i).add(true);
            west.get(i).add(true);
            south.get(i).add(true);
            visited.get(i).add(false);
        }
    }
```

Figur 1: Initialisering av listor

Vi använder oss av 2-dimensionella boolean listor eftersom med hjälp av dessa kan vi ha listor som tar in en koordinat i form av ett x och y värde sedan säga om det finns en vägg i given riktning. Efter dessa finns det även en femte lista som säger om algoritmen har varit på denna cell innan, detta behövs så inte algoritmen skriver över tidigare “tunnlar”

Efter initialiseringen kallas den första av två **generate** metoder. Den första har inga parametrar och körs bara en gång för att göra mer förberedelser, viktigast är att en stack kallade **mazeStack** instansieras, denna behövs för algoritmen när den ska backtracka.
Sedan slumpas ett x och y värde fram som startposition innan den andra **generate** metoden kallas med start positionens x och y som parametrar. Algoritmen visas i figur 2.

I figuren visas hur algoritmen bestämmer vilket håll den ska gå, först slumpas riktningen fram och om algoritmen inte redan varit på den cellen så tas väggarna bort mellan de två cellerna och

metoden kallas rekursivt från den nya cellen. Loopen används för att säkerställa att en ny giltig riktning blir given om den slumpade riktningen redan var besökt. Om ingen riktning är giltig kommer den gå ur loopen och poppa av den senaste värdet från stacken, därefter kommer **generate** bli kallad från den tidigare noden. Detta är vad som ger algoritmen backtracking.

```
while(true)
{
    Random rand = new Random();
    int p = rand.nextInt(4);
    if(p == 0 && !visited.get(x).get(y+1)) {

        //Går uppåt
        mazeStack.push(currentNode);
        north.get(x).set(y, false);
        south.get(x).set(y+1, false);
        draw(currentNode);
        generate(x, y + 1);
        break;
    }
```

Figur 2: Ett exempel av algoritmen när den väljer att gå uppåt

MazeGenSideWind

Den andra algoritmen för att generera labyrinter är en simpel sidewinder som går radvis och genererar labyrinter med horisontella föredrag. Den använder likadan instansiering av listor som den tidigare algoritmen så denna förklaring börjar direkt med själva algoritmen.
Algoritmen börjar alltid i det övre vänstra hörnet av kvadraten. Först och främst måste hela den översta raden vara en korridor, detta eftersom det inte går att gå upp från högsta raden.
Sedan kommer den börja från den andra raden med att alltid gå ett steg till höger, efter detta är det en slump om den väljer att fortsätta ett steg till höger eller skära ett steg uppåt, Algoritmen visas i figur 3.
Varje gång algoritmen går ett steg till höger läggs den cellen in i en lista kallad **currentSet.** Sedan när algoritmen slumpmässigt väljer att skära uppåt så kommer den slumpmässigt välja en cell i denna lista och skära uppåt. Detta i praktiken skapar en labyrint gjord av flera horisontella korridorer. Se figur i resultat för exempel.
carveEast metoden går ett steg till höger och tar bort väggarna mellan de två cellerna, medan **carveNorth** är mer komplex och visas i figur 4.

```
public void generate(int x, int y) {
    if(!done) {
        if(y == 0)
        {
            done = true;
            return;
        }
        if(x == n+1 || x == n) {

            if(currentSet.size() == 0){
                north.get(x).set(y, false);
                south.get(x).set(y+1, false);
                generate(1,y-1);
            }
            else {
                carveNorth(x,y);
            }
        }
        else {
            boolean goEast = goOrStop();

            if(currentSet.size() == 0 || goEast) {
                carveEast(x,y);
            }
            else{
                carveNorth(x,y);
            }
```

Figur 3: MazeGenSideWind generate metod

När **carveNorth** blir kallad används **currentSet** för att veta vilka celler denna korridor uppstår av. Efter att metoden har skurit en väg uppåt så rensas listan och nästa korridor kan skapas.

```
public void carveNorth(int x, int y) {

    Random random = new Random();
    int passageIndex = random.nextInt(currentSet.size());
    north.get(currentSet.get(passageIndex).getX()).set(y, false);
    south.get(currentSet.get(passageIndex).getX()).set(y+1, false);

    currentSet.clear();
    if(x == n) {
        generate(1,y-1);
    }
    else {
        generate(x+1,y);
    }
```

Figur 4: MazeGenSideWind carveNorth metod

ImportedMaze

Den tredje typen av labyrint generation är importerade. För att importera används en textfil med koordinater för varje cell och dess boolean värden för väggar samma som tidigare labyrinter. För att läsa in används en scanner och en loop för att tilldela varje koordinat sitt värde. Ett exempel på en labyrint fil syns i figur 5 och metod för inläsning syns i figur 6. Ordningen av booleans går som väderstreck: Nord,Öst,Syd,Väst.

```
1,1 true false true true
1,2 false false true true
1,3 false true false true
1,4 false true false true
1,5 false true false true
1,6 false true false true
1,7 false true false true
```

Figur 5: Textfil som innehåller data för labyrint

```
else{
    int x = Integer.parseInt(dataReader.next());
    int y = Integer.parseInt(dataReader.next());
    maze.getNorth().get(x).set(y,Boolean.parseBoolean(dataReader.next()));
    maze.getEast().get(x).set(y,Boolean.parseBoolean(dataReader.next()));
    maze.getSouth().get(x).set(y,Boolean.parseBoolean(dataReader.next()));
    maze.getWest().get(x).set(y,Boolean.parseBoolean(dataReader.next()));
```

Figur 6: Metod för att läsa in data från tidigare textfil

Labyrintlösning

Algoritmer för de båda lösningsalogoritmer använder sig båda av fyra 2-dimensionella boolean listor för att kunna traversera de genererade labyrinterna. De båda implementerar våran MazeSolver interface för att de ska kunna användas av användaren beroende på vad den vill välja.

DepthFirst

Denna rekursiva sökalgoritm använder sig av som sagt en lista med 2-dimensionella booleans, de är riktningar. Det finns även en för besökta positioner.
Med hjälp av väggarna som finns i labyrinten kollar den om det är tillåtet att gå åt någon riktning beroende på om det är en vägg där. Det är även en regel att den inte får gå till positioner den redan har varit på för att det x och y värdet är markerad besökt om algoritmen har vart där en gång tidigare. Med hjälp av if-satser så görs detta, men att gå upp är högsta prioritet medans att gå åt höger kommer därefter som visas på figur 7.
Denna algoritm är klar när det förvalda värdet av användaren som i detta fall är 'n' har nåtts av både x och y värdet.

```
if (!north.get(x).get(y)) {
    solve(x, y + 1);
}
if (!east.get(x).get(y)) {
    solve(x + 1, y);
}
if (!south.get(x).get(y)) {
    solve(x, y - 1);
}
if (!west.get(x).get(y)) {
    solve(x - 1, y);
}
```

Figur 7: Depth First rekursiv sök metod

Stack loop

Denna sökalgoritm har också som sagt likadan lista med riktningar men använder sig av de på ett annat sätt. Denna algoritm använder sig av en stack och noder. Där även upp och höger har den högsta prioriteten.

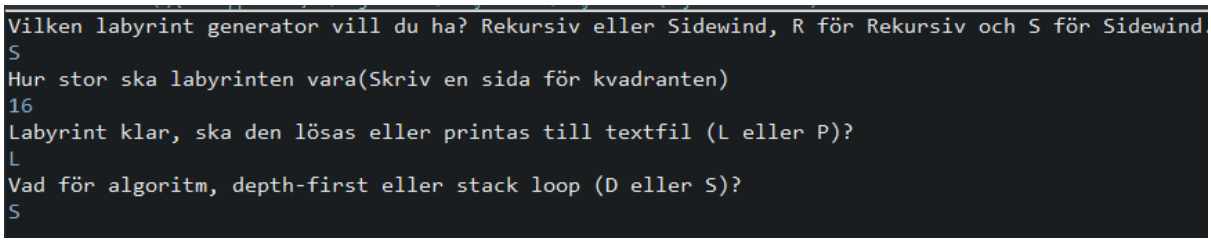
```
else if (x + 1 <= n && !east.get(x).get(y) && !visited.get(x+1).get(y)) {
    // Försöker flytta höger
    MazeNode thisNode = new MazeNode(x,y);
    visited.get(x).set(y,true);
    stack.push(thisNode);
    x++;
}
else if (y + 1 <= n && !north.get(x).get(y) && !visited.get(x).get(y+1)) {
    // Försöker flytta up
    MazeNode thisNode = new MazeNode(x,y);
    visited.get(x).set(y,true);
    stack.push(thisNode);
    y++;
}
else if (y - 1 > 0 && !south.get(x).get(y) && !visited.get(x).get(y-1)) {
    // Försöker flytta nedåt
    MazeNode thisNode = new MazeNode(x,y);
    visited.get(x).set(y,true);
    stack.push(thisNode);
    y--;
}
else if (x - 1 > 0 && !west.get(x).get(y) && !visited.get(x-1).get(y)) {
    // Försöker flytta vänster
    MazeNode thisNode = new MazeNode(x,y);
    visited.get(x).set(y,true);
    stack.push(thisNode);
    x--;
}
```

Figur 8: Stack loop iterativ sök metod

I figur 8 så visar den de olika 'checks' som är implementerad. Här visas att flytta höger är viktigare än att gå upp jämfört med den tvärtomt rekursiva lösningen. Dessa 'checks' kollar om villkoret stämmer vilket är om den kan fortsätta i riktningar i denna prioritetsordning höger-upp-vänster-nedåt. Det som finns inom villkoren är att värdet åt höger t.ex inte finns i stacken eller har redan blivit besökt. Om det stämmer så pushas den nuvarande positionen till stacken. Den positionen blir då märkt besökt och värdet för x inkrementeras för att positionen har flyttat ett steg åt höger, de andra 'checks' fungerar på samma sätt fast för sina respektive riktningar.

Main

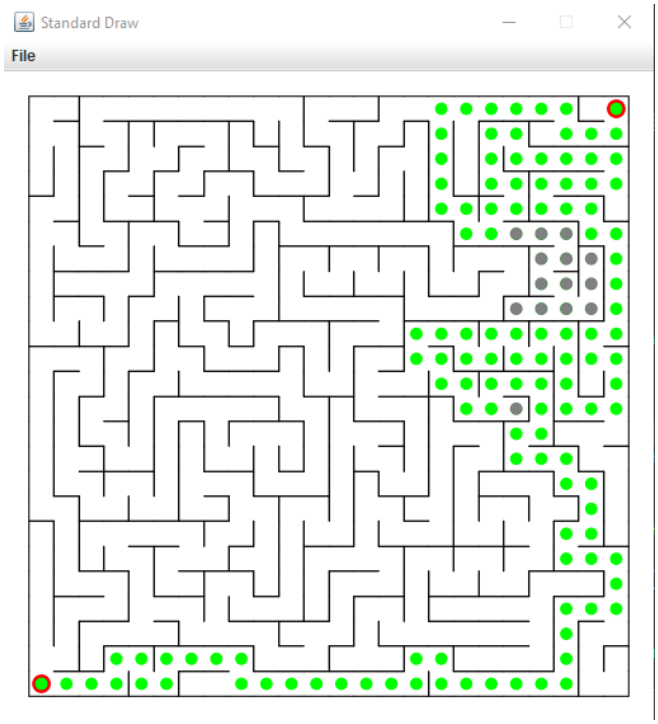
I main så bestäms användningen av dessa nämnda labyrinter och sökalgoritmer, som en kontrollpanel. Användaren blir frågad om hur den vill köra programmet. I figur 9 så visas ett exempel på en exekvering av main.



Figur 9: Nuvarande användargränssnitt är enbart en konsol med frågor.

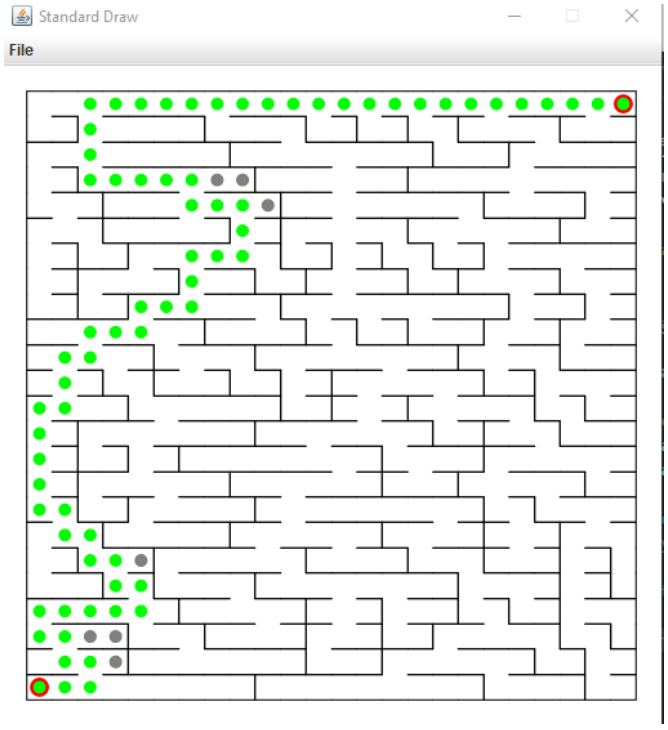
Resultat

I figur 10 så visas den rekursiva backtrack labyrinten som är löst med hjälp av Stack loop sökalgoritmen.



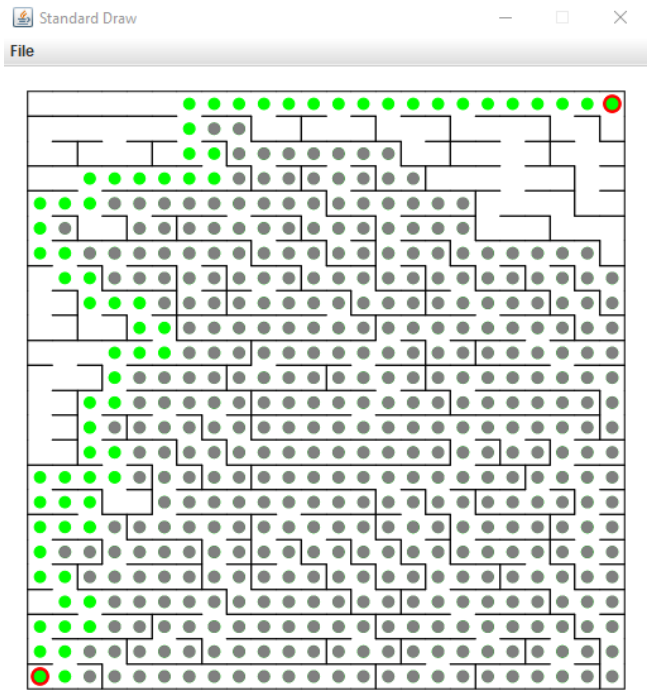
Figur 10: Rekursiv labyrint med stack loop sökning

I figur 11 så visas Sidewind labyrinten som löses med hjälp av den rekursiva DepthFirst sökalgoritmen.



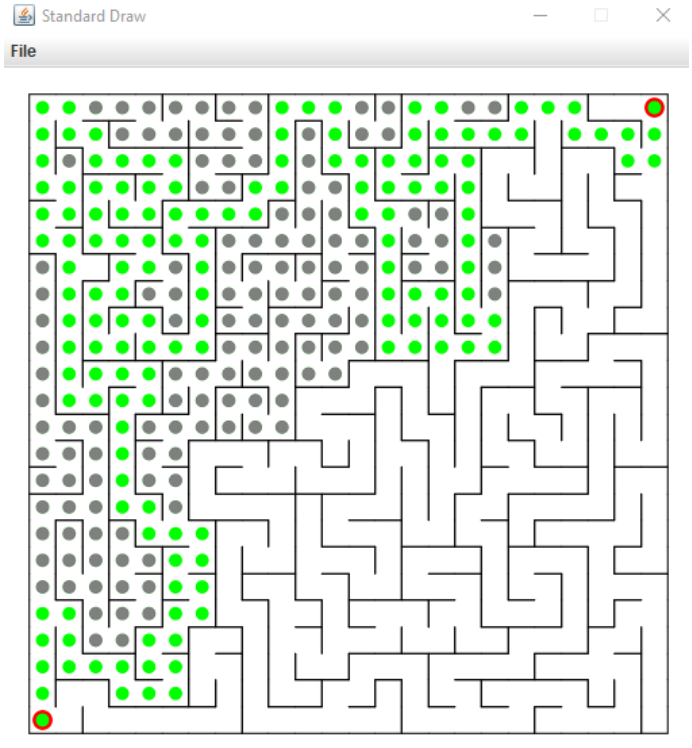
Figur 11: SideWind labyrint med DepthFirst sökning

I figur 12 så visas Sidewind labyrinten som är löst med hjälp av Stack loop sökalgoritmen



Figur 12: SideWind labyrint med StackLoop sökning

I figur 13 så visas den rekursiva backtrack labyrinten som är löst med hjälp av den rekursiva Depth First sökalgoritmen



Figur 13: Rekursiv labyrint med DepthFirst sökning

Diskussion

Det största problemet som vi stötte på var att hitta bra algoritmer för generering av våra labyrinter, det gick åt en del tid för algoritmer som vi i slutändan gick vidare ifrån. De var antingen inte bra eller så fungerade de inte som vi hade planerat och vi kände att det skulle ta för mycket tid att lösa det när det blev väldigt svårt att förstå. Men till slut hittade vi algoritmer vi var nöjd över som även kunde använda sig av samma 2-dimensionella boolean listor. När de båda typer av labyrint generering var klar så var det lätta kvar. Vi letade snabbt upp olika sätt för att lösa en labyrint och vi valde en rekursiv lösning och en som använder sig av stack. De var lättare att koda för då fanns det redan någonting att arbeta mot, de genererade labyrinterna.

Att använda prioriteten av riktning till upp och höger var på grund av att vi startar från x,y(1,1). För att snabbast komma till x,y(24,24) så går man ett steg upp och sen höger i repetitiv ordning för att komma till det högsta x och y värdet så snabbt som möjligt som våran labyrint är genererad.

Projektet var en lärande upplevelse men stressad på grund av andra studieuppgifter. Det gav oss mindre tid för arbetet och förbättringar har kunnat gjorts. Den förlorade tiden gör också att denna rapport är påverkad negativt.

Våran Sidewind labyrint är extremt lätt att lösa genom att prioritera uppflyttning upp och höger kommer i andra prioritet. Det visar figur ? på, där den rekursiva sökalgoritmen DepthFirst löser labyrinten utan knappt några fel. Vilket är positivt men hade vart roligare om alla labyrinter var avancerad.

Appendix

ImportedMaze

```
1.  package mazeGeneration;
2.  /**
3.   * @author William Tiderman
4.   * @author John Engblom Sandin
5.   * @version 2021-01-10
6.   */
7.  import java.util.List;
8.
9.
10.
11.         public class ImportedMaze implements MazeGeneration {
12.
13.             private int n;
14.             // 2-Dimensionella listor som säger om det finns en vägg åt ett
    håll från positionen
15.             private List<List<Boolean>> north;
16.             private List<List<Boolean>> east;
17.             private List<List<Boolean>> west;
18.             private List<List<Boolean>> south;
19.             private List<List<Boolean>> visited;
20.
21.             public ImportedMaze(List<List<Boolean>>
    north,List<List<Boolean>> east,List<List<Boolean>>
    south,List<List<Boolean>> west,List<List<Boolean>> visited, int n) {
22.                 this.north = north;
23.                 this.east = east;
24.                 this.south = south;
25.                 this.west = west;
26.                 this.visited = visited;
27.                 this.n = n;
28.             }
29.
```

```
30.         @Override
31.         public void draw() {
32.             // TODO Auto-generated method stub
33.
34.         }
35.
36.         @Override
37.         public List<List<Boolean>> getNorth() {
38.             // TODO Auto-generated method stub
39.             return this.north;
40.         }
41.
42.         @Override
43.         public List<List<Boolean>> getEast() {
44.             // TODO Auto-generated method stub
45.             return this.east;
46.         }
47.
48.         @Override
49.         public List<List<Boolean>> getWest() {
50.             // TODO Auto-generated method stub
51.             return this.west;
52.         }
53.
54.         @Override
55.         public List<List<Boolean>> getSouth() {
56.             // TODO Auto-generated method stub
57.             return this.south;
58.         }
59.
60.         @Override
61.         public int getN() {
62.             // TODO Auto-generated method stub
63.             return this.n;
64.         }
65.
66.         @Override
67.         public void generate() {
68.             // TODO Auto-generated method stub
69.
```

```
70.             }
71.
72.             @Override
73.             public void generate(int x, int y) {
74.                 // TODO Auto-generated method stub
75.
76.             }
77.
78.             @Override
79.             public List<List<Boolean>> getVisited() {
80.                 // TODO Auto-generated method stub
81.                 return this.visited;
82.             }
83.
84.         }
```

Main

```
1.  package mazeGeneration;
2.  /**
3.   * @author William Tiderman
4.
5.   * @author John Engblom Sandin
6.   * @version 2021-01-10
7.   */
8.  import java.util.Scanner;
9.
10.         public class Main {
11.
12.             /**
13.              * Frågar användaren om vilken labyrint och lösningsalgoritm
    som ska köras.
14.              *
15.              * @param args
16.              */
17.             public static void main(String[] args) {
18.                 MazeGeneration maze = null;
```

```
19.             MazeSolver mazeSolver = null;
20.
21.             Scanner choose = new Scanner(System.in);
22.             System.out.println("Vilken labyrint generator vill du ha? "
23.                     + "Rekursiv eller Sidewind, R för Rekursiv och S för
    Sidewind. Annars skriv I om du vill importa från textfil");
24.             String yesNo = choose.nextLine();
25.
26.             if(yesNo.equalsIgnoreCase("I")) {
27.                 System.out.println("Skriv in den exakt sökvägen till
    textfilen");
28.                 String filePath = choose.nextLine();
29.
30.                 MazeImport mazeImport = new MazeImport();
31.
32.                 maze = mazeImport.importMaze(filePath);
33.             }
34.             else {
35.                 System.out.println("Hur stor ska labyrinten vara(Skriv en
    sida för kvadranten)");
36.                 String size = choose.nextLine();
37.
38.                 int n = 15;
39.                 try {
40.                     n = Integer.parseInt(size);
41.                 } catch (NumberFormatException e) {
42.                     System.out.println("Inte ett nummer");
43.                     e.printStackTrace();
44.                 }
45.
46.                 if(yesNo.equalsIgnoreCase("R")) {
47.                     maze = new MazeGenRecBack(n);
48.                     StdDraw.enableDoubleBuffering();
49.                 }
50.                 else if(yesNo.equalsIgnoreCase("S")) {
51.                     maze = new MazeGenSideWind(n);
52.                     StdDraw.enableDoubleBuffering();
53.                 }
54.                 else if(!yesNo.equalsIgnoreCase("r") ||
    !yesNo.equalsIgnoreCase("s")){
```

```
55.                     System.out.println("Fel inmatning av val");
56.                 }
57.             }
58.
59.             System.out.println("Labyrint klar, ska den lösas eller printas
    till textfil (L eller P)?");
60.             String solveOrPrint = choose.nextLine();
61.
62.             if(solveOrPrint.equalsIgnoreCase("L")) {
63.                 System.out.println("Vad för algoritm, depth-first eller stack
    loop (D eller S)?");
64.                 String solve = choose.nextLine();
65.
66.                 if(solve.equalsIgnoreCase("D")) {
67.                     mazeSolver = new MazeSolveDepthFirst(maze);
68.                 }
69.                 if(solve.equalsIgnoreCase("S")) {
70.                     mazeSolver = new MazeSolveLoop(maze);
71.                 }
72.                 mazeSolver.solve();
73.
74.             }
75.             else if(solveOrPrint.equalsIgnoreCase("P")) {
76.                 MazeExport export = new MazeExport(maze);
77.                 System.out.println("Print klar");
78.             }
79.             else if(!solveOrPrint.equalsIgnoreCase("P") ||
    !solveOrPrint.equalsIgnoreCase("L")){
80.                 System.out.println("Fel inmatning av val");
81.             }
82.
83.
84.             choose.close();
85.         }
86.
87.     }
```

MazeExport

```
1.  package mazeGeneration;
2.  /**
3.   * @author William Tiderman
4.   * @author John Engblom Sandin
5.   * @version 2021-01-10
6.   */
7.  import java.io.FileWriter;
8.  import java.io.IOException;
9.  import java.util.List;
10.
11.         public class MazeExport {
12.
13.             // 2-Dimensionella listor som säger om det finns en vägg åt ett
    håll från positionen
14.             private List<List<Boolean>> north;
15.             private List<List<Boolean>> east;
16.             private List<List<Boolean>> west;
17.             private List<List<Boolean>> south;
18.             private int n;
19.
20.             /**
21.              * Exporterar en labyrint till en textfil.
22.              *
23.              * @param maze
24.              */
25.
26.             public MazeExport(MazeGeneration maze) {
27.
28.                 this.n = maze.getN();
29.                 this.north = maze.getNorth();
30.                 this.east = maze.getEast();
31.                 this.west = maze.getWest();
32.                 this.south = maze.getSouth();
33.                 try {
34.                     String outPrint = "Första två siffrorna är koordinater sedan
    true eller false om det finns en vägg i riktningen
    Norr,Öst,Söder,Västerut \n";
```

```
                    outPrint += String.valueOf(n)+ "\n";

                    for(int i = 1; i <= n;i++){
                        for(int j = 1; j <= n; j++){

                            outPrint += i + "," + j + " " + north.get(i).get(j) + " " +
    east.get(i).get(j) + " " + south.get(i).get(j) + " " + west.get(i).get(j) + "\n";

                        }
                    }

                    FileWriter myWriter = new FileWriter("MazeFile.txt");
                    myWriter.write(outPrint);
                    myWriter.close();

                } catch (IOException e) {
                    System.out.println("Det blev fel med skrivningen");
                    e.printStackTrace();
                }
            }
        }
```

MazeGeneration

```
1.  package mazeGeneration;
2.  /**
3.   * @author William Tiderman
4.   * @author John Engblom Sandin
5.   * @version 2021-01-10
6.   */
7.  import java.util.List;
8.
9.  /**
10.          * Interface för skapning av labyrint.
11.          *
12.          */
13.         public interface MazeGeneration {
```

```
14.
15.             public void draw();
16.             public List<List<Boolean>> getNorth();
17.             public List<List<Boolean>> getEast();
18.             public List<List<Boolean>> getWest();
19.             public List<List<Boolean>> getSouth();
20.             int getN();
21.             public void generate();
22.             public void generate(int x, int y);
23.             public List<List<Boolean>> getVisited();
24.
25.         }
```

MazeGenRecBack

```
1.  package mazeGeneration;
2.  /**
3.   * @author William Tiderman
4.   * @author John Engblom Sandin
5.   * @version 2021-01-10
6.   */
7.  import java.util.ArrayList;
8.  import java.util.List;
9.  import java.util.Random;
10.         import java.util.Stack;
11.
12.
13.         public class MazeGenRecBack implements MazeGeneration{
14.
15.             // 2-Dimensionella listor som säger om det finns en vägg åt ett
    håll från positionen
16.             private int n;
17.             private List<List<Boolean>> north;
18.             private List<List<Boolean>> east;
19.             private List<List<Boolean>> west;
20.             private List<List<Boolean>> south;
21.             private List<List<Boolean>> visited;
```

```
22.         Stack<MazeNode> mazeStack;
23.         MazeNode startNode;
24.
25.
26.         /**
27.          * Ritar ut en röd cirkel för startpunk och slutpunkt samt skapar
    labyrintens väggar
28.          *
29.          * @param currentNode, hämtar värdet för vilket X och Y som
    är noden för alla riktningar
30.          */
31.         public void draw(MazeNode currentNode) {
32.             StdDraw.setPenColor(StdDraw.RED);
33.             StdDraw.filledCircle(n + 0.5, n + 0.5, 0.375);
34.             StdDraw.filledCircle(1.5, 1.5, 0.375);
35.
36.             StdDraw.setPenColor(StdDraw.BLACK);
37.
38.             if (south.get(currentNode.getX()).get(currentNode.getY()))
    StdDraw.line(currentNode.getX(), currentNode.getY(),
    currentNode.getX()+1, currentNode.getY());
39.             if (north.get(currentNode.getX()).get(currentNode.getY()))
    StdDraw.line(currentNode.getX(), currentNode.getY()+1,
    currentNode.getX()+1, currentNode.getY()+1);
40.             if (west.get(currentNode.getX()).get(currentNode.getY()))
    StdDraw.line(currentNode.getX(), currentNode.getY(),
    currentNode.getX(), currentNode.getY()+1);
41.             if (east.get(currentNode.getX()).get(currentNode.getY()))
    StdDraw.line(currentNode.getX()+1, currentNode.getY(),
    currentNode.getX()+1, currentNode.getY()+1);
42.         }
43.
44.         /**
45.          * Ritar ut röd cirkel för slut och starpunkt samt ritar labyrintens
    väggar ut
46.          */
47.         public void draw(){
48.             StdDraw.clear();
49.             StdDraw.setPenColor(StdDraw.RED);
50.             StdDraw.filledCircle(n + 0.5, n + 0.5, 0.375);
```

```
51.             StdDraw.filledCircle(1.5, 1.5, 0.375);
52.             StdDraw.setPenColor(StdDraw.BLACK);
53.             for (int x = 1; x <= n; x++) {
54.                 for (int y = 1; y <= n; y++) {
55.                     if (south.get(x).get(y)) StdDraw.line(x, y, x+1, y);
56.                     if (north.get(x).get(y)) StdDraw.line(x, y+1, x+1, y+1);
57.                     if (west.get(x).get(y))  StdDraw.line(x, y, x, y+1);
58.                     if (east.get(x).get(y))  StdDraw.line(x+1, y, x+1, y+1);
59.                 }
60.             }
61.
62.             StdDraw.show();
63.         }
64.
65.         public List<List<Boolean>> getNorth(){
66.             return this.north;
67.         }
68.         public List<List<Boolean>> getEast(){
69.             return this.east;
70.         }
71.         public List<List<Boolean>> getWest(){
72.             return this.west;
73.         }
74.         public List<List<Boolean>> getSouth(){
75.             return this.south;
76.         }
77.         public List<List<Boolean>> getVisited(){
78.             return this.visited;
79.         }
80.
81.         public int getN() {
82.             return n;
83.         }
84.
85.         // Initialisation för listorna och bestämma att alla väggar är uppe
86.         private void init() {
87.             north = new ArrayList<List<Boolean>>();
88.             east = new ArrayList<List<Boolean>>();
89.             west = new ArrayList<List<Boolean>>();
90.             south = new ArrayList<List<Boolean>>();
```

```
91.             visited = new ArrayList<List<Boolean>>();
92.
93.             for(int i = 0; i < n+2; i++){
94.                 north.add(new ArrayList<Boolean>());
95.                 east.add(new ArrayList<Boolean>());
96.                 west.add(new ArrayList<Boolean>());
97.                 south.add(new ArrayList<Boolean>());
98.                 visited.add(new ArrayList<Boolean>());
99.
100.                for(int j = 0; j < n+2; j++) {
101.                    north.get(i).add(true);
102.                    east.get(i).add(true);
103.                    west.get(i).add(true);
104.                    south.get(i).add(true);
105.                    visited.get(i).add(false);
106.                }
107.            }
108.        }
109.
110.        /**
111.         * Generar labyrinten
112.         */
113.        public void generate() {
114.            mazeStack = new Stack<MazeNode>();
115.
116.            for (int x = 0; x < n+2; x++) {
117.                visited.get(x).set(0,true);
118.                visited.get(x).set(n+1,true);
119.            }
120.            for (int y = 0; y < n+2; y++) {
121.                visited.get(0).set(y,true);
122.                visited.get(n+1).set(y,true);
123.            }
124.
125.            Random rand = new Random();
126.            int randX = rand.nextInt(n-1) +1;
127.            int randY = rand.nextInt(n-1) +1;
128.
129.            startNode = new MazeNode(randX, randY);
130.
```

```
131.            generate(randX,randY);
132.
133.            draw();
134.        }
135.
136.        public MazeGenRecBack(int n) {
137.            this.n = n;
138.            StdDraw.setXscale(0, n+2);
139.            StdDraw.setYscale(0, n+2);
140.            init();
141.            generate();
142.        }
143.
144.        /**
145.         * Generar labyrint med hjälp av rekursiv backtracking
146.         *
147.         * @param x, värdet för x axeln
148.         * @param y, värdet för y axeln
149.         */
150.        public void generate(int x, int y) {
151.            MazeNode currentNode = new MazeNode(x,y);
152.
153.            visited.get(x).set(y, true);
154.
155.            if(!visited.get(x).get(y+1) || !visited.get(x).get(y-1) ||
    !visited.get(x+1).get(y) || !visited.get(x-1).get(y)) {
156.
157.                while(true)
158.                {
159.                    Random rand = new Random();
160.                    int p = rand.nextInt(4);
161.                    if(p == 0 && !visited.get(x).get(y+1)) {
162.
163.                        //Går uppåt
164.                        mazeStack.push(currentNode);
165.                        north.get(x).set(y, false);
166.                        south.get(x).set(y+1, false);
167.                        draw(currentNode);
168.                        generate(x, y + 1);
169.                        break;
```

```
170.                 }
171.                 if(p == 1 && !visited.get(x).get(y-1)) {
172.
173.                     //Går neråt
174.                     mazeStack.push(currentNode);
175.                     south.get(x).set(y, false);
176.                     north.get(x).set(y-1, false);
177.                     draw(currentNode);
178.                     generate(x, y - 1);
179.                     break;
180.                 }
181.                 if(p == 2 && !visited.get(x+1).get(y)) {
182.
183.                     //Går åt höger
184.                     mazeStack.push(currentNode);
185.                     east.get(x).set(y, false);
186.                     west.get(x+1).set(y, false);
187.                     draw(currentNode);
188.                     generate(x+1, y);
189.                     break;
190.                 }
191.                 if(p == 3 && !visited.get(x-1).get(y)) {
192.
193.                     //Går åt vänster
194.                     mazeStack.push(currentNode);
195.                     west.get(x).set(y, false);
196.                     east.get(x-1).set(y, false);
197.                     draw(currentNode);
198.                     generate(x-1, y);
199.                     break;
200.                 }
201.             }
202.             MazeNode lastNode = mazeStack.pop();
203.
204.             if(lastNode.getX() == startNode.getX() && lastNode.getY()
    == startNode.getY()){
205.                 return;
206.             }
207.             else{
208.                 generate(lastNode.getX(),lastNode.getY());
```

```
209.                 }
210.             }
211.         }
212.     }
```

MazeGenSideWind

```
1.  package mazeGeneration;
2.  /**
3.   * @author William Tiderman
4.   * @author John Engblom Sandin
5.   * @version 2021-01-10
6.   */
7.  import java.util.ArrayList;
8.  import java.util.List;
9.  import java.util.Random;
10.
11.         public class MazeGenSideWind implements MazeGeneration {
12.
13.             // 2-Dimensionella listor som säger om det finns en vägg åt ett
    håll från positionen
14.             private int n;
15.             private List<List<Boolean>> north;
16.             private List<List<Boolean>> east;
17.             private List<List<Boolean>> west;
18.             private List<List<Boolean>> south;
19.             private List<List<Boolean>> visited;
20.
21.             private List<MazeNode> currentSet;
22.             private boolean done = false;
23.
24.
25.             @Override
26.             public List<List<Boolean>> getNorth() {
27.                 // TODO Auto-generated method stub
28.                 return this.north;
29.             }
```

```
30.
31.         @Override
32.         public List<List<Boolean>> getEast() {
33.             // TODO Auto-generated method stub
34.             return this.east;
35.         }
36.
37.         @Override
38.         public List<List<Boolean>> getWest() {
39.             // TODO Auto-generated method stub
40.             return this.west;
41.         }
42.
43.         @Override
44.         public List<List<Boolean>> getSouth() {
45.             // TODO Auto-generated method stub
46.             return this.south;
47.         }
48.         @Override
49.         public List<List<Boolean>> getVisited() {
50.             // TODO Auto-generated method stub
51.             return this.visited;
52.         }
53.
54.         @Override
55.         public int getN() {
56.             return this.n;
57.         }
58.
59.         /**
60.          * Skapar layout för labyrint
61.          *
62.          * @param n dimension för kvadraten av labyrinten
63.          */
64.         public MazeGenSideWind(int n) {
65.             this.n = n;
66.             StdDraw.setXscale(0, n+2);
67.             StdDraw.setYscale(0, n+2);
68.             init();
69.             generate();
```

```
70.         }
71.
72.         /**
73.          * Ritar start/slut punkt samt väggar i labyrinten
74.          */
75.         @Override
76.         public void draw() {
77.             StdDraw.clear();
78.             StdDraw.pause(10);
79.
80.             StdDraw.setPenColor(StdDraw.RED);
81.             StdDraw.filledCircle(n + 0.5, n + 0.5, 0.375);
82.             StdDraw.filledCircle(1.5, 1.5, 0.375);
83.             StdDraw.setPenColor(StdDraw.BLACK);
84.             for (int x = 1; x <= n; x++) {
85.                 for (int y = 1; y <= n; y++) {
86.                     if (south.get(x).get(y)) StdDraw.line(x, y, x+1, y);
87.                     if (north.get(x).get(y)) StdDraw.line(x, y+1, x+1, y+1);
88.                     if (west.get(x).get(y))  StdDraw.line(x, y, x, y+1);
89.                     if (east.get(x).get(y))  StdDraw.line(x+1, y, x+1, y+1);
90.                 }
91.             }
92.
93.             StdDraw.show();
94.
95.         }
96.
97.         //init
98.         private void init() {
99.             north = new ArrayList<List<Boolean>>();
100.            east = new ArrayList<List<Boolean>>();
101.            west = new ArrayList<List<Boolean>>();
102.            south = new ArrayList<List<Boolean>>();
103.            visited = new ArrayList<List<Boolean>>();
104.
105.            currentSet = new ArrayList<MazeNode>();
106.
107.            for(int i = 0; i < n+2; i++){
108.                north.add(new ArrayList<Boolean>());
109.                east.add(new ArrayList<Boolean>());
```

```
110.                west.add(new ArrayList<Boolean>());
111.                south.add(new ArrayList<Boolean>());
112.                visited.add(new ArrayList<Boolean>());
113.
114.                for(int j = 0; j < n+2; j++) {
115.                    north.get(i).add(true);
116.                    east.get(i).add(true);
117.                    west.get(i).add(true);
118.                    south.get(i).add(true);
119.                    visited.get(i).add(true);
120.                }
121.            }
122.        }
123.        @Override
124.        public void generate() {
125.
126.            for (int x = 1; x <= n; x++) {
127.                north.get(x).set(n,true);
128.                south.get(x).set(1,true);
129.            }
130.            for (int y = 1; y <= n; y++) {
131.                east.get(n).set(y,true);
132.                west.get(1).set(y,true);
133.            }
134.
135.
136.            east.get(1).set(n, false);
137.
138.            for(int i = 2; i < n; i++) {
139.                west.get(i).set(n, false);
140.                east.get(i).set(n, false);
141.            }
142.
143.            west.get(n).set(n, false);
144.
145.            generate(1,n-1);
146.
147.            draw();
148.
149.
```

```
150.        }
151.
152.        @Override
153.        public void generate(int x, int y) {
154.            if(!done) {
155.                if(y == 0)
156.                {
157.                    done = true;
158.                    return;
159.                }
160.                if(x == n+1 || x == n) {
161.
162.                    if(currentSet.size() == 0){
163.                        north.get(x).set(y, false);
164.                        south.get(x).set(y+1, false);
165.                        generate(1,y-1);
166.                    }
167.                    else {
168.                        carveNorth(x,y);
169.                    }
170.                }
171.                else {
172.                    boolean goEast = goOrStop();
173.
174.                    if(currentSet.size() == 0 || goEast) {
175.                        carveEast(x,y);
176.                    }
177.                    else{
178.                        carveNorth(x,y);
179.                    }
180.                }
181.            }
182.        }
183.        /**
184.         * Randomizar och den ska gå åt höger eller uppåt
185.         *
186.         * @return boolean för höger eller upp
187.         */
188.        public boolean goOrStop() {
189.
```

```
190.            Random random = new Random();
191.            int ifgo = random.nextInt(3-1)+1;
192.
193.            if(ifgo < 2) {
194.                return true;
195.            }
196.            else {
197.                return false;
198.            }
199.        }
200.        /**
201.         * Skär väggen till höger och kallar rekursivt på generate igen
202.         *
203.         * @param x värdet på x koordinaten
204.         * @param y värdet på y koordinaten
205.         */
206.        public void carveEast(int x, int y) {
207.            MazeNode currentNode = new MazeNode(x,y);
208.            currentSet.add(currentNode);
209.            east.get(x).set(y, false);
210.            west.get(x+1).set(y, false);
211.            generate(x+1,y);
212.        }
213.        /**
214.         * Skär väggen till höger och kallar rekursivt på generate igen
215.         *
216.         * @param x värdet på x koordinaten
217.         * @param y värdet på y koordinaten
218.         */
219.        public void carveNorth(int x, int y) {
220.
221.            Random random = new Random();
222.            int passageIndex = random.nextInt(currentSet.size());
223.            north.get(currentSet.get(passageIndex).getX()).set(y, false);
224.            south.get(currentSet.get(passageIndex).getX()).set(y+1,
    false);
225.
226.            currentSet.clear();
227.            if(x == n) {
228.                generate(1,y-1);
```

```
229.                }
230.                else {
231.                    generate(x+1,y);
232.                }
233.
234.
235.            }
236.        }
```

MazeImport

```
1.  package mazeGeneration;
2.  /**
3.   * @author William Tiderman
4.   * @author John Engblom Sandin
5.   * @version 2021-01-10
6.   */
7.  import java.io.File;
8.  import java.io.FileNotFoundException;
9.  import java.util.ArrayList;
10.         import java.util.List;
11.         import java.util.Scanner;
12.
13.         public class MazeImport{
14.
15.             // 2-Dimensionella listor som säger om det finns en vägg åt ett
    håll från positionen
16.             private int n;
17.             private List<List<Boolean>> north;
18.             private List<List<Boolean>> east;
19.             private List<List<Boolean>> west;
20.             private List<List<Boolean>> south;
21.             private List<List<Boolean>> visited;
22.             MazeGeneration maze = null;
23.
24.             public MazeImport() {
25.             }
```

```
26.
27.         public MazeGeneration importMaze(String filePath) {
28.             String data = "";
29.
30.             north = new ArrayList<List<Boolean>>();
31.             east = new ArrayList<List<Boolean>>();
32.             west = new ArrayList<List<Boolean>>();
33.             south = new ArrayList<List<Boolean>>();
34.             visited = new ArrayList<List<Boolean>>();
35.
36.
37.
38.
39.             File textFile = new File(filePath);
40.             Scanner mazeFile = null;
41.             try {
42.                 mazeFile = new Scanner(textFile);
43.             } catch (FileNotFoundException e) {
44.                 // TODO Auto-generated catch block
45.                 e.printStackTrace();
46.             }
47.
48.             while (mazeFile.hasNextLine())
49.             {
50.                 data += mazeFile.nextLine();
51.                 data += "\n";
52.             }
53.             data = data.replaceAll(","," ");
54.             Scanner dataReader = new Scanner(data);
55.             dataReader.useDelimiter(" |\n");
56.
57.
58.             boolean firstRun = true;
59.
60.             while(dataReader.hasNext())
61.             {
62.                 if(firstRun) {
63.                     firstRun = false;
64.                     dataReader.nextLine();
65.
```

```
66.                     n = Integer.parseInt(dataReader.next());
67.                     for(int i = 0; i < n+2; i++){
68.                         north.add(new ArrayList<Boolean>());
69.                         east.add(new ArrayList<Boolean>());
70.                         west.add(new ArrayList<Boolean>());
71.                         south.add(new ArrayList<Boolean>());
72.                         visited.add(new ArrayList<Boolean>());
73.
74.                         for(int j = 0; j < n+2; j++) {
75.                             north.get(i).add(true);
76.                             east.get(i).add(true);
77.                             west.get(i).add(true);
78.                             south.get(i).add(true);
79.                             visited.get(i).add(false);
80.                         }
81.                     }
82.                     maze = new
    ImportedMaze(north,east,south,west,visited,n);
83.                 }
84.                 else{
85.                     int x = Integer.parseInt(dataReader.next());
86.                     int y = Integer.parseInt(dataReader.next());
87.
    maze.getNorth().get(x).set(y,Boolean.parseBoolean(dataReader.next())
    );
88.
    maze.getEast().get(x).set(y,Boolean.parseBoolean(dataReader.next()));
89.
    maze.getSouth().get(x).set(y,Boolean.parseBoolean(dataReader.next())
    );
90.
    maze.getWest().get(x).set(y,Boolean.parseBoolean(dataReader.next()))
    ;
91.
92.                     String systemPrint = x + "," + y + " " + north.get(x).get(y)
    + " " + east.get(x).get(y) + " " + south.get(x).get(y) + " " +
    west.get(x).get(y)+ visited.get(x).get(y) + "\n";
93.                     System.out.println(systemPrint);
94.                 }
95.
```

```
96.             }
97.             draw();
98.             dataReader.close();
99.             return maze;
100.        }
101.
102.        public void draw() {
103.
104.            StdDraw.setXscale(0, n+2);
105.            StdDraw.setYscale(0, n+2);
106.            StdDraw.clear();
107.            StdDraw.pause(10);
108.
109.            StdDraw.setPenColor(StdDraw.RED);
110.            StdDraw.filledCircle(n + 0.5, n + 0.5, 0.375);
111.            StdDraw.filledCircle(1.5, 1.5, 0.375);
112.            StdDraw.setPenColor(StdDraw.BLACK);
113.            for (int x = 1; x <= n; x++) {
114.                for (int y = 1; y <= n; y++) {
115.                    if (south.get(x).get(y)) StdDraw.line(x, y, x+1, y);
116.                    if (north.get(x).get(y)) StdDraw.line(x, y+1, x+1, y+1);
117.                    if (west.get(x).get(y))  StdDraw.line(x, y, x, y+1);
118.                    if (east.get(x).get(y))  StdDraw.line(x+1, y, x+1, y+1);
119.                }
120.            }
121.
122.            StdDraw.show();
123.
124.        }
125.    }
```

MazeNode

```
1.  package mazeGeneration;
2.  /**
3.   * @author William Tiderman
```

```
4.   * @author John Engblom Sandin
5.   * @version 2021-01-10
6.   */
7.  public class MazeNode {
8.      private int x;
9.      private int y;
10.
11.             private int endX;
12.             private int endY;
13.
14.             private boolean north;
15.             private boolean east;
16.             private boolean west;
17.             private boolean south;
18.             private boolean visited;
19.
20.
21.             public int getY() {
22.                 return y;
23.             }
24.             public void setY(int y) {
25.                 this.y = y;
26.             }
27.             public int getX() {
28.                 return x;
29.             }
30.             public void setX(int x) {
31.                 this.x = x;
32.             }
33.
34.             public MazeNode(int x, int y) {
35.                 this.x = x;
36.                 this.y = y;
37.             }
38.
39.             public boolean isNorth() {
40.                 return north;
41.             }
42.             public void setNorth(boolean north) {
43.                 this.north = north;
```

```
44.         }
45.         public boolean isEast() {
46.             return east;
47.         }
48.         public void setEast(boolean east) {
49.             this.east = east;
50.         }
51.         public boolean isWest() {
52.             return west;
53.         }
54.         public void setWest(boolean west) {
55.             this.west = west;
56.         }
57.         public boolean isSouth() {
58.             return south;
59.         }
60.         public void setSouth(boolean south) {
61.             this.south = south;
62.         }
63.         public boolean isVisited() {
64.             return visited;
65.         }
66.         public void setVisited(boolean visited) {
67.             this.visited = visited;
68.         }
69.         public int getEndY() {
70.             return endY;
71.         }
72.         public void setEndY(int endY) {
73.             this.endY = endY;
74.         }
75.         public int getEndX() {
76.             return endX;
77.         }
78.         public void setEndX(int endX) {
79.             this.endX = endX;
80.         }
81.
82.
83.
```

```
84.         }
```

MazeSolveDepthFirst

```
1.  package mazeGeneration;
2.  /**
3.   * @author William Tiderman
4.   * @author John Engblom Sandin
5.   * @version 2021-01-10
6.   */
7.  import java.util.List;
8.
9.  public class MazeSolveDepthFirst implements MazeSolver {
10.
11.             // 2-Dimensionella listor som säger om det finns en vägg åt ett
    håll från positionen
12.             private int n;   // dimension of maze
13.             private List<List<Boolean>> north;
14.             private List<List<Boolean>> east;
15.             private List<List<Boolean>> west;
16.             private List<List<Boolean>> south;
17.             private List<List<Boolean>> visited;
18.             private boolean done = false;
19.
20.             public MazeSolveDepthFirst(MazeGeneration mazeGen){
21.                 this.n = mazeGen.getN();
22.                 this.north = mazeGen.getNorth();
23.                 this.east = mazeGen.getEast();
24.                 this.west = mazeGen.getWest();
25.                 this.south = mazeGen.getSouth();
26.                 this.visited = mazeGen.getVisited();
27.
28.
29.             }
30.
31.             // Rekursiv metod för att lösa labyrinten med hjälp av en depth-
    first algoritm
```

```
32.         public void solve(int x, int y) {
33.
34.             if (x == 0 || y == 0 || x == n+1 || y == n+1) {
35.                 return;
36.             }
37.
38.             if (done || visited.get(x).get(y)) {
39.                 return;
40.             }
41.             visited.get(x).set(y,true);
42.
43.             StdDraw.setPenColor(StdDraw.GREEN);
44.             StdDraw.filledCircle(x + 0.5, y + 0.5, 0.25);
45.             StdDraw.show();
46.             StdDraw.pause(20);
47.
48.             // reached Ending
49.             if (x == n && y == n) done = true;
50.
51.
52.             if (!north.get(x).get(y)) {
53.                 solve(x, y + 1);
54.             }
55.             if (!east.get(x).get(y)) {
56.                 solve(x + 1, y);
57.             }
58.             if (!south.get(x).get(y)) {
59.                 solve(x, y - 1);
60.             }
61.             if (!west.get(x).get(y)) {
62.                 solve(x - 1, y);
63.             }
64.
65.             if (done) return;
66.
67.             StdDraw.setPenColor(StdDraw.GRAY);
68.             StdDraw.filledCircle(x + 0.5, y + 0.5, 0.25);
69.             StdDraw.show();
70.             StdDraw.pause(20);
71.         }
```

```
72.
73.             // Kallar på solve metoden från startpunkten.
74.             public void solve() {
75.                 for (int x = 1; x <= n; x++)
76.                     for (int y = 1; y <= n; y++)
77.                         visited.get(x).set(y, false);
78.                 done = false;
79.                 solve(1, 1);
80.             }
81.         }
```

MazeSolveLoop

```
1.  package mazeGeneration;
2.  /**
3.   * @author William Tiderman
4.   * @author John Engblom Sandin
5.   * @version 2021-01-10
6.   */
7.  import java.util.List;
8.  import java.util.Stack;
9.
10.         public class MazeSolveLoop implements MazeSolver {
11.
12.             // 2-Dimensionella listor som säger om det finns en vägg åt ett
    håll från positionen
13.             private int n;
14.             private List<List<Boolean>> north;
15.             private List<List<Boolean>> east;
16.             private List<List<Boolean>> south;
17.             private List<List<Boolean>> west;
18.             private List<List<Boolean>> visited;
19.             private boolean done = false;
20.             Stack<MazeNode> stack;
21.
22.             public MazeSolveLoop(MazeGeneration mazeGen) {
23.                 this.n = mazeGen.getN();
```

```
24.             this.north = mazeGen.getNorth();
25.             this.east = mazeGen.getEast();
26.             this.west = mazeGen.getWest();
27.             this.south = mazeGen.getSouth();
28.             this.visited = mazeGen.getVisited();
29.         }
30.         public void solve(int x, int y) {
31.             stack = new Stack<MazeNode>();
32.
33.             StdDraw.setPenColor(StdDraw.BLUE);
34.             StdDraw.filledCircle(x + 0.5, y + 0.5, 0.25);
35.             StdDraw.show();
36.             StdDraw.pause(20);
37.
38.             visited.get(1).set(1,true);
39.             MazeNode startNode = new MazeNode(1,1);
40.             stack.push(startNode);
41.
42.             while (true) {
43.                 draw(true,x,y);
44.
45.                 if (x == n && y == n) {
46.                     // Hittade slutpunkten
47.                     done = true;
48.                 }
49.                 else if (x + 1 <= n && !east.get(x).get(y) &&
    !visited.get(x+1).get(y)) {
50.                     // Försöker flytta höger
51.                     MazeNode thisNode = new MazeNode(x,y);
52.                     visited.get(x).set(y,true);
53.                     stack.push(thisNode);
54.                     x++;
55.                 }
56.                 else if (y + 1 <= n && !north.get(x).get(y) &&
    !visited.get(x).get(y+1)) {
57.                     // Försöker flytta up
58.                     MazeNode thisNode = new MazeNode(x,y);
59.                     visited.get(x).set(y,true);
60.                     stack.push(thisNode);
61.                     y++;
```

```
62.                 }
63.                 else if (y - 1 > 0 && !south.get(x).get(y) &&
    !visited.get(x).get(y-1)) {
64.                     // Försöker flytta nedåt
65.                     MazeNode thisNode = new MazeNode(x,y);
66.                     visited.get(x).set(y,true);
67.                     stack.push(thisNode);
68.                     y--;
69.                 }
70.                 else if (x - 1 > 0 && !west.get(x).get(y) && !visited.get(x-
    1).get(y)) {
71.                     // Försöker flytta vänster
72.                     MazeNode thisNode = new MazeNode(x,y);
73.                     visited.get(x).set(y,true);
74.                     stack.push(thisNode);
75.                     x--;
76.                 }
77.
78.                 else if (!stack.isEmpty()) {
79.
80.                     visited.get(x).set(y,true);
81.                     draw(false,x,y);
82.
83.                     MazeNode lastNode = stack.pop();
84.                     x = lastNode.getX();
85.                     y = lastNode.getY();
86.                 }
87.                 else {
88.                     // Ingen utväg hittades
89.                     break;
90.                 }
91.                 if (done) return;
92.             }
93.         }
94.         /**
95.          * Ritar ut en blå en grå cirkel på noden
96.          *
97.          * @param blue ett sant/falskt värde för att veta vilken färg som
    ska ritas
98.          * @param x nodens x värde
```

```
99.              * @param y nodens y värde
100.             */
101.            public void draw(boolean blue, int x, int y) {
102.                if(blue) {
103.                    StdDraw.setPenColor(StdDraw.GREEN);
104.                }
105.                else {
106.                    StdDraw.setPenColor(StdDraw.GRAY);
107.                }
108.                StdDraw.filledCircle(x + 0.5, y + 0.5, 0.25);
109.                StdDraw.setPenColor(StdDraw.RED);
110.                StdDraw.show();
111.                StdDraw.pause(20);
112.            }
113.
114.
115.            public void solve() {
116.                for (int x = 1; x <= n; x++)
117.                    for (int y = 1; y <= n; y++)
118.                        visited.get(x).set(y, false);
119.                done = false;
120.                solve(1,1);
121.            }
122.        }
```

MazeSolver

```
1.  package mazeGeneration;
2.  /**
3.   * @author William Tiderman
4.   * @author John Engblom Sandin
5.   * @version 2021-01-10
6.   */
7.
8.
9.  public interface MazeSolver {
10.
```

```
11.             void solve();
12.             void solve(int x, int y);
13.
14.         }
```