Syddansk Universitet
IMADA

06-11-2020
DS801

INTRODUKTION TIL OBJEKTORIENTERET PROGRAMMERING:

Projekt 1/3 (Gruppe 25)

Koordinator: Sarah Pedersen [redacted]
Simon Lorentzen [redacted]
Stefan Randa ([redacted]

Indhold

1. Indledning

Denne rapport dokumenterer den første fase i eksamensprojektet til kurset *Introduktion til objektorienteret programmering* (DS801). Kravene til projektet er specificeret i det kommende afsnit. Derefter beskrives nogle af de designbeslutninger, som gruppen har taget, og derpå uddybes implementeringen af disse beslutninger. Til sidst foreligger et afsnit om testscenarier, en konklusion og relevante bilag.

2. Kravspecifikation

Gruppen skal i denne fase af projektet udarbejde en kildefil, *RunSimulation.java*, som er en "top-level class", der igangsætter simulationen ved hjælp af følgende brugerdefinerede parametre:

- Sandsynligheden for, at en given knude i grafen starter med at indeholde sukker.
- Den gennemsnitlige mængde af sukker i sådan en knude.
- Mængden af sukker, som en myre kan bære.
- Mængden af feromoner, som en myre udskyder, når den rejser igennem en knude.
- Den samlede mængde af kolonier i simulationen.
- Mængden af myrer i hver koloni.
- Om grafen skal genereres tilfældigt eller ud fra en fil, og i denne forbindelse:
    - En brugerdefineret længde og bredde, som grafen skal genereres ud fra.
    - Eller navnet på filen, som grafen skal genereres ud fra.
- Hvor længe simulationen skal køre.
- Om brugeren vil se simulationen grafisk eller opsummeret som tekst.

Samt en rapport, *report.pdf*, som inkluderer:

- En beskrivelse af og eksempler på den implementerede kode.
- Relevante designbeslutninger.
- Scenarier, metoder og resultater fra test af koden.
- Et bilag, som inkluderer kildekoden.

Disse to filer skal afleveres i én samlet ZIP-fil uden undermapper.

3. Design

"RunSimulation"-klassen er designet ud fra den udleverede dokumentation og generel brugervenlighed.[1] I den forbindelse har gruppen bl.a. truffet følgende designbeslutninger:

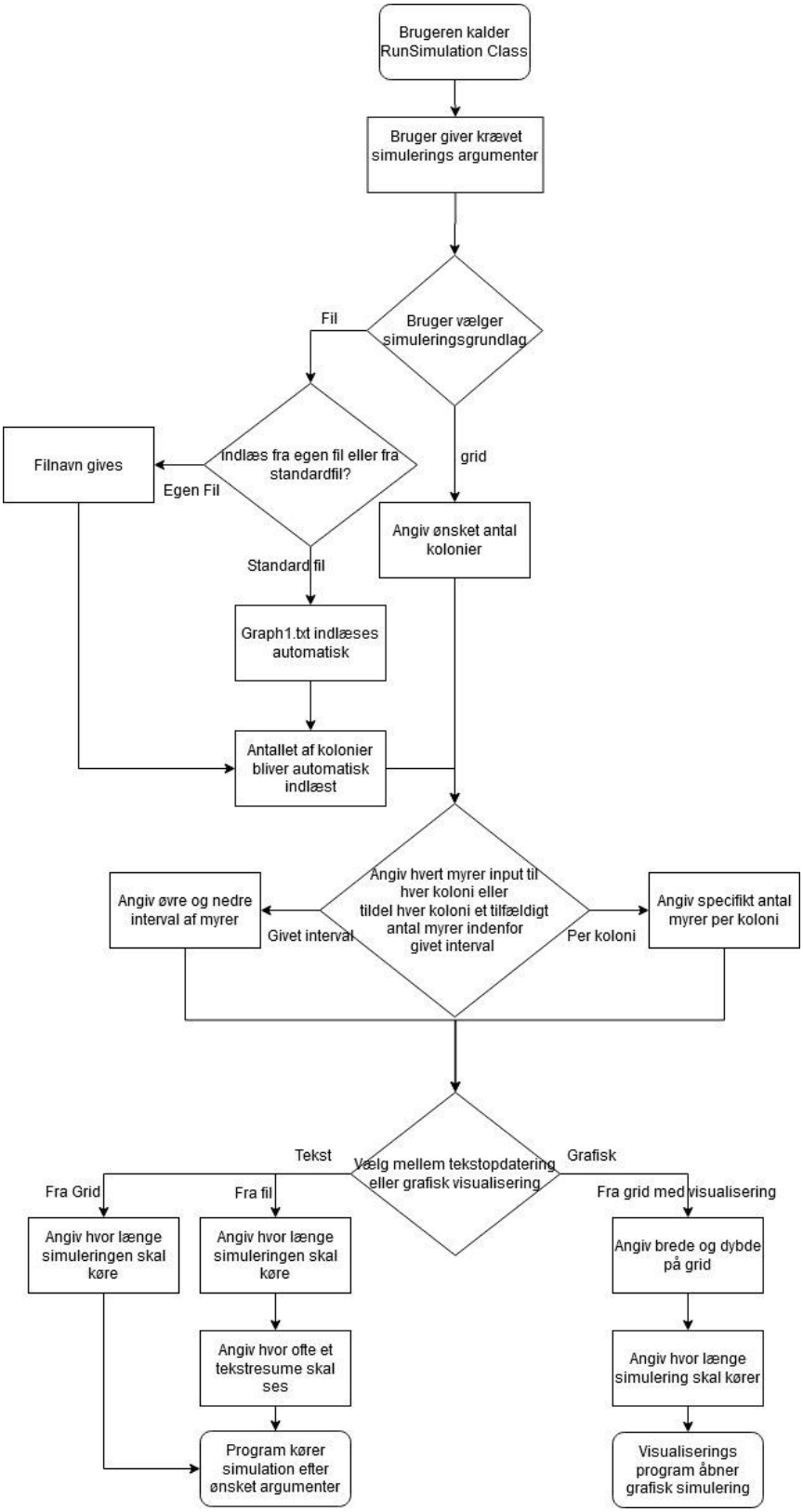
- Alle brugerefterspurgte argumenter eksekveres via argumenter som "Please enter [...]", "How do you wish to [...]" etc.

- Der er udviklet en valideringslogik, som sikrer rette argumenter fra brugeren, så programmet ikke nødvendigvis fejler ved et forkert argument.

- For at imødekomme forskellige scenarier, kan kolonierne både genereres automatisk og manuelt. Myrer kan både gives til programmet automatisk i intervaller eller manuelt per koloni. Således undgås et scenarie, hvor en bruger f.eks. efterspørger 100 kolonier, men ikke ønsker at angive 100 forskellige argumenter.

- Brugeren skal foretage en række valg gennem opsætningen af simulationen. Dette håndteres ved, at brugeren vælger enten A eller B i programmet. I programmet er der endvidere indarbejdet en funktionalitet, som sikrer at brugeren indtaster valide valgmuligheder.

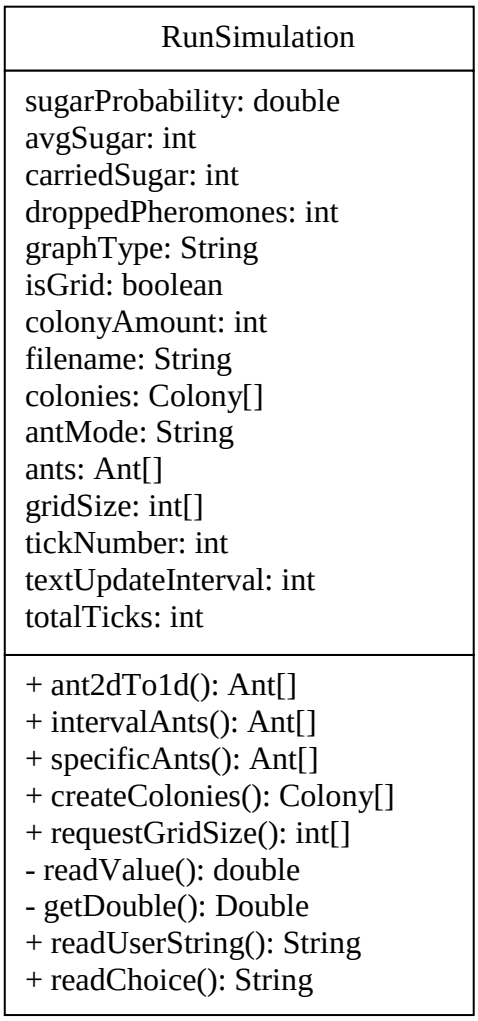
[1] For en visualisering af brugerrejsen i programmet, se procesdiagrammet i *Figur 1*.



Figur 1. Procesdiagram.

4. Implementering

I dette afsnit foreligger en kort uddybning af de mest essentielle metoder, som er udviklet til "RunSimulation"-klassen. Det følgende UML-inspirerede klassediagram (*Figur 2*) giver et hurtigt overblik over, hvad klassen indeholder:



Figur 2. Klassediagram.

4.1. readValue()-metoden

Metoden "readValue()" er en "double", som tager en instans af "Scanner" og en "String", som printes ud til konsollen. Denne metode er skrevet for at gøre indhentningen af information fra brugeren nemmere og for at tjekke, om de brugerindtastede informationer også har en brugbar talværdi. Dette gør test og fejlfinding lettere i udviklingsprocessen.

"readValue()" printer først den besked, som er specificeret i "String input", og afventer derefter en indtastet talværdi fra brugeren. Så testes den indtastede værdi for at se, om det er

en gyldig "double", og hvis det ikke er tilfældet, bedes brugeren om at indtaste en ny værdi. Når al indtastet data er gennemgået, returnerer metoden den indtastede og gyldige værdi.

```
private static double readValue(Scanner scanner, String message, double definition) {
    System.out.println(message);
    String input = scanner.nextLine();
    while (input != null) {
        Double value = null;
        if (input.equalsIgnoreCase("default")) {
            break;
        }
        if (input.trim().isEmpty() || (value = getDouble(input)) == null) {
            System.out.println(input + " is not a valid input. Try again!");
            System.out.println(message);
        }
        if (value != null) {
            return value;
        }
        if (scanner.hasNextLine()) {
            input = scanner.nextLine();
        } else {
            input = null;
        }
    }
    System.out.println("Using default value of " + definition);
    return definition;
}
```

4.2. ant2dTo1d()-metoden

Metoden "ant2dTo1d()" er udviklet for at imødekomme nogle af de udfordringer, som kan opstå ved genereringen af kolonier og myrer.

Kolonier danner grundlag for, at myrer kan oprettes, eftersom de er det eneste element, der indgår i "Ant"-konstruktøren. I "RunSimulation"-klassen er der to muligheder for at angive antal kolonier: kolonier kan indlæses automatisk via en fil (efter specifikationerne fra "Graph"-klassens dokumentation), eller brugeren kan angive kolonierne manuelt. Hvordan kolonier skal gives til variablen, afgøres, når brugeren vælger, om grafen skal genereres fra Grid eller fra tekstfil.[2] Når antallet af kolonier er angivet, bliver et koloni-array oprettet. Denne løsning kræver, at kolonier og myrer oprettes ved at kalde klasserne, og at instanserne gemmes i et array. Derfor valgte gruppen at bruge et 2D-array og iterationer som løsning, eftersom dette ville nedbringe mængden af variabler – i stedet for, at myrer og kolonier f.eks. blev lagt i et array hver for sig. I bilaget kan metoderne "intervalAnts()" og "specificAnts()" ses. Begge metoder benytter sig af "ant2dTo1d()"-metoden, der gør, at metoderne kan returnere et 1D array, som indgår i konstruktørerne for "Visualizer" og "Simulator".

[2] For en visualisering af denne proces, se atter procesdiagrammet i *Figur 1*.

```
// loop through simulation
int totalTicks = 0;
while (totalTicks < tickNumber) {
    if (viewMode.equalsIgnoreCase("B")) {
        visualizer.update();
    } else if (totalTicks % textUpdateInterval == 0) {
        visualizer.printStatus();
    }
    simulator.tick();
    totalTicks = totalTicks + 1;
}
```

Ovenfor ses kodens motor. Efter brugeren har angivet alle argumenter til de påkrævede parametre for opsætningen af simulationen, skal ticks angives, så simulationen kan køre. Argumentet gemmes i variablen "tickNumber". For at køre simulationen, laves et while loop, som enten kalder "update()"- eller "printStatus()"-metoden. Loopet løber gennem iterationer, og kalder "tick()"-metoden, indtil "totalTicks" har samme værdi, som det angivne "tickNumber". Uden dette stykke kode, kan simulationen ikke eksekveres.

5. Test

Gruppen har løbende testet koden, både som enkelte algoritmer og hele klassen. Tilgangen har været manuel såvel som "automatiseret" ved at indlæse et dokument med argumenter til hver variable.[3] I tabellen på næste side ses en liste over de endelige testscenarier for funktionalitet. Kolonnen "Rettet" beskriver, hvorvidt udfaldet skal rettes eller ej; "N/A" indikerer, at der ikke arbejdes videre med det scenarie.

Case 3, 4 og 9 havde et andet udfald end forventet. Case 3 og 9 accepterer negative argumenter, hvor programmet fortsætter og åbner simulationen. Case 4 accepterer, at der kan være 0 kolonier i programmet, men det stopper dog, når simulationsvinduet er eksekveret. I case 4 bliver der i programmet tilføjet et forbehold for indtastningen af 0 kolonier, eftersom det er en af kravspecifikationerne. Derfor vil brugeren i fremtiden se en fejlbesked ved indtastning af 0 kolonier. Case 3 og 9 vil forblive som de er nu, idet gruppen arbejder ud fra kontraktbaseret programmering.

[3] Se *Bilag II* for TestMedGrid.txt.

Dato	Case	Emne	Testscenarie	Forventet resultat	Resultat	Rettet
3/11/2020	1	Argumenter fra bruger til første 4 parameter	Valide argumenter bliver gemt i variabler	Ja	Programmet kører som forventet	N/A
3/11/2020	2	Argumenter fra bruger til første 4 parameter	Programmet returnerer fejlbesked ved indtastning af forkerte argumenter, f.eks. M & !	Viser fejlbesked	Som forventet	N/A
3/11/2020	3	Argumenter fra bruger til første 4 parameter	Programmet returnerer fejlbesked ved negative værdier	Programmet stopper	Program modtager negative værdier	Nej
3/11/2020	4	Kolonier	Bruger indtaster 0 kolonier	Fejlbesked vises til bruger	Program accepterer 0 kolonier	Ja
3/11/2020	5	Kolonier	Bruger indtaster 1 eller flere kolonier	Argumentet bliver modtaget	Som forventet	N/A
3/11/2020	6	Myrer	Bruger indtaster 0 myrer som argument	Argumentet bliver modtaget	Som forventet	N/A
3/11/2020	7	Myrer	Bruger kan angive myrer ved brug af interval-funktion	Argumenter bliver modtaget	Som forventet	N/A
3/11/2020	8	Myrer	Bruger kan angive myrer ved brug af per koloni-funktion	Argumenter bliver modtaget	Som forventet	N/A

3/11/2020	9	Myrer	Bruger giver myrer-funktioner negative argumenter	Programmet stopper	Negativt argument modtages	Nej
3/11/2020	10	Myrer	Bruger indtaster bogstaver eller symboler	Fejlbesked vises	Som forventet	N/A
3/11/2020	11	Fil	Bruger vælger fil og trykker enter	Besked om at kolonier læses automatisk fra graph1.txt vises	Som forventet	N/A
3/11/2020	12	Fil	Bruger giver et fejl input, f.eks. “b”	Program stopper med at virke	Som forventet	N/A
3/11/2020	13	Fil	Efter indlæsning af fil vælger brugeren at se simulationen i tekst	Simulation kører	Som forventet	N/A
3/11/2020	14	Grid	Bruger indtaster Grid < 3*3	Fejlbesked vises	Som forventet	N/A
3/11/2020	15	Grid	Indtastet Grid dimensioner er mindre end antallet af kolonier	Fejlbesked vises	Som forventet	N/A
3/11/2020	16	Grid	Bruger giver intet input, men trykker ENTER	Fejlbesked vises	Som forventet	N/A
3/11/2020	17	Grid	Grafisk simulation kan køre med Grid	Simulation kører	Som forventet	N/A
3/11/2020	18	Grid	Tekst simulation kan køre med Grid	Simulation kører	Som forventet	N/A

6. Konklusion

Kravspecifikationen til programmet såvel som rapporten er opfyldt. Brugeren kan køre *RunSimulation.java* direkte fra sin kommandolinjegrænseflade, hvor alle relevante argumenter og informationer vil blive indsamlet og fremvist. Design og implementering er forklaret, og programmets funktionalitet er testet gennem flere scenarier. Derfor vurderes det, at koden er funktionel, og at programmet efterlever kravspecifikationerne.

Referencer

Foruden den udleverede klassedokumentation, har gruppen konsulteret lærebogen efter behov:

- Savitch, W. (2019). *An Introduction to Problem Solving & Programming*. Harlow, UK: Pearson Education Limited.

Bilag I (RunSimulation.java)

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

/*
 * This is the top-level class for the Ant Colony Simulation.
 */
public class RunSimulation {

    // Call Scanner for later access
    static Scanner scanner;

    /*
     * This method executes the simulation, allowing for exceptions to be thrown.
     */
    public static void main(String[] args) throws FileNotFoundException {

        // Create new instance of Scanner
        scanner = new Scanner(System.in);

        // Welcome
        System.out.println("");
        System.out.println("Welcome to the Ant Colony Simulation.");
        System.out.println("");

        // Get user inputs
        double sugarProbability = readValue(scanner,
                "What is the probability of sugar spawning in a node? Please enter number: ");
        System.out.println("The sugar probability is " + sugarProbability + ".");
        System.out.println("");

        int avgSugar = (int) readValue(scanner, "What is the average amount of sugar in a node? Please enter number: ");
        System.out.println("The average sugar amount is " + avgSugar + " units.");
        System.out.println("");

        int carriedSugar = (int) readValue(scanner,
                "How many units of sugar can ants carry at a time? Please enter number: ");
        System.out.println("Ants can carry " + carriedSugar + " units of sugar.");

        System.out.println("");

        int droppedPheromones = (int) readValue(scanner, "How many pheromones do ants secrete? Please enter number: ");
        System.out.println("Ants secrete " + droppedPheromones + " pheromones.");
        System.out.println("");

        String graphType = readChoice(scanner,
                "How do you wish to generate the graph? \n"
                        + "- By reading a file storing the graph, press and enter 'A'. \n"
                        + "- By creating your own grid, press and enter 'B'. \n",
                "A", "B");

        boolean isGrid = graphType.equalsIgnoreCase("B");

        // Define variables due to scoping
        int colonyAmount;
        String filename = "Filename not found.";

        // Manual colony user input
        if (graphType.equalsIgnoreCase("B")) {
            while (true) {
                colonyAmount = (int) readValue(scanner, "Please type the total number of colonies: ");
                if (colonyAmount > 0) {
                    System.out.println("Total number of Ant Colonies is: " + colonyAmount);
                    break;
                }
                System.out.println("The total number of colonies must be at least 1 ");
            }
        }
        // Create grid and generate colonies based on file input
        else {
            filename = readUserString(scanner, "Please enter filename. Otherwise, graph1.txt is chosen.", "graph1.txt");

            // Read second line from graph file
            Scanner filescanner = new Scanner(new File(filename));
            filescanner.nextLine();
            colonyAmount = filescanner.nextLine().trim().split(" ").length;
            System.out.println("The number of colonies is " + colonyAmount + ".");

        }

        Colony[] colonies = createColonies(colonyAmount);

        String antMode = readChoice(scanner,
                "How do you wish to assign the ants? \n" + "- By range of ants per colony, press and enter 'A'. \n"
                        + "- By specific number of ants per colony, press and enter 'B'. \n",
                "A", "B");
        System.out.println("");

        Ant[] ants;
        // Assign ants in intervals
        if (antMode.equalsIgnoreCase("A")) {
            ants = intervalAnts(colonies);
        }
        // Assign ants per colony
        else {
```

```
            ants = specificAnts(colonies);
        }

        // Let user choose mode of viewing information
        String viewMode;
        if (isGrid) {
            viewMode = readChoice(scanner,
                    "Please enter the desired mode of viewing information. \n"
                            + "- For a textual summary, press and enter 'A'. \n"
                            + "- For a graphical representation, press and enter 'B'. \n",
                    "A", "B");
            System.out.println("");
        } else {
            viewMode = "A";
        }

        // Declare instance of Graph in proper scope
        Graph graph;

        // Read file from disk
        if (graphType.equalsIgnoreCase("A")) {
            System.out.println("The graph is generated by the file " + filename + ".");

            // Create new instance of Graph instance
            graph = new Graph(filename, colonies, sugarProbability, avgSugar);

            // Specify width and height of new grid based on user input
        } else {
            int[] gridSize = requestGridSize(colonyAmount);
            int width = gridSize[0];
            int depth = gridSize[1];
            graph = new Graph(width, depth, colonies, sugarProbability, avgSugar);
            System.out.println("The dimensions of your grid are " + width + "x" + depth + ".");
        }

        int tickNumber = (int) readValue(scanner, "How many simulations do wish to run?");
        int textUpdateInterval = 1;

        if (viewMode.equalsIgnoreCase("A")) {
            textUpdateInterval = (int) readValue(scanner, "How often do you wish to receive a text update in ticks?");
        }

        // Create new instance of Simulator
        Simulator simulator = new Simulator(graph, ants, carriedSugar, droppedPheromones);

        // Get start node from ant array
        Node startNode = ants[0].current();
        System.out.println("StartNode:" + startNode);

        // Create new instance of Visualizer
        Visualizer visualizer = new Visualizer(graph, isGrid, startNode, ants);

        if (viewMode.equalsIgnoreCase("B")) {
            visualizer.display();
        }

        // Loop through simulation
        int totalTicks = 0;
        while (totalTicks < tickNumber) {
            if (viewMode.equalsIgnoreCase("B")) {
                visualizer.update();
            } else if (totalTicks % textUpdateInterval == 0) {
                visualizer.printStatus();
            }
            simulator.tick();
            totalTicks = totalTicks + 1;
        }

    }

    /*
     * This method converts the Ant 2D array into the Ant 1D array. Argument: Ant 2D
     * array. Returns Ant 1D array.
     */
    public static Ant[] ant2dTo1d(Ant[][] ant2d) {
        int x = 0;
        int sum = 0;
        while (x < ant2d.length) {
            sum = sum + ant2d[x].length;
            x = x + 1;
        }
        Ant[] ant1d = new Ant[sum];
        x = 0;
        int i = 0;
        while (x < ant2d.length) {
            int y = 0;
            while (y < ant2d[x].length) {
                ant1d[i] = ant2d[x][y];
                i = i + 1;
                y = y + 1;
            }
            x = x + 1;
        }
        return ant1d;
    }

    /*
     * This method allows the user to assign ants automatically in a lower and upper
     * range. Argument: Colony 1D array. Returns Ant 1D array in desired interval.
```

```
 * In case of wrong input, the user must enter a new argument.
 */
public static Ant[] intervalAnts(Colony[] colonies) {
    int lowerRange = (int) readValue(scanner, "Please enter lower range: ");
    System.out.println("");
    int upperRange = (int) readValue(scanner, "Please enter upper range: ");
    while (upperRange < lowerRange) {
        System.out.println("Upper range must be larger than or equal to lower range!");
        upperRange = (int) readValue(scanner, "Please enter upper range: ");
    }
    int n = upperRange - lowerRange;
    int x = 0;
    Ant[][] ants = new Ant[colonies.length][];
    // Generate random number to ant array per colony
    while (x < colonies.length) {
        int randomRange = RandomUtils.randomInt(n) + lowerRange;
        Ant[] antsArr = new Ant[randomRange];
        int y = 0;
        // Generate ants
        while (y < antsArr.length) {
            Ant ant = new Ant(colonies[x]);
            antsArr[y] = ant;
            y = y + 1;
        }
        ants[x] = antsArr;
        x = x + 1;
    }
    return ant2dTo1d(ants);
}

/*
 * This method allows the user to assign ants specifically for each colony.
 * Argument: Colony 1D array. Returns Ant 1D array.
 */
public static Ant[] specificAnts(Colony[] colonies) {
    System.out.println("Assign ants for each of the " + colonies.length + "colonies:");
    int x = 0;
    Ant[][] ants = new Ant[colonies.length][];
    System.out.println("");
    while (x < colonies.length) {
        int antsAmount = (int) readValue(scanner, "Enter ants for colony:" + (x + 1));
        Ant[] antsArr = new Ant[antsAmount];
        int y = 0;

        while (y < antsArr.length) {
            Ant ant = new Ant(colonies[x]);
            antsArr[y] = ant;
            y = y + 1;

        }
        ants[x] = antsArr;
        x = x + 1;
    }
    return ant2dTo1d(ants);
}

/*
 * This method generates colonies and assigns it to the colony array. Argument:
 * int colony amount. Return Colony 1D array.
 */
public static Colony[] createColonies(int colonyAmount) {
    Colony[] colonies = new Colony[colonyAmount];
    int i = 0;
    while (i < colonies.length) {
        Colony colony = new Colony();
        colonies[i] = colony;
        i = i + 1;
    }
    return colonies;
}

/*
 * This method allows the user to assign grid arguments. Argument: int colony
 * amount. Returns int 1D array with grid arguments. In case of invalid input
 * (i.e. < 3*3), the user must enter a new argument.
 */
public static int[] requestGridSize(int colonyAmount) {
    while (true) {
        System.out.println("Please define grid size (the dimensions must be at least 3x3).");
        int width = (int) readValue(scanner, "Enter grid width: ");
        int depth = (int) readValue(scanner, "Enter grid height: ");
        int gridSize = width * depth;
        if (width < 3 || depth < 3) {
            continue;
        } else if (gridSize < colonyAmount) {
            System.out.println("The amount of colonies is too large for grid!");
        } else {
            return new int[] { width, depth };
        }
    }
}

/*
 * This method reads and checks values from the user. Arguments: Scanner and
 * string message. Returns a message and default value to the user.
 */
private static double readValue(Scanner scanner, String message) {
    return readValue(scanner, message, 0.0);
}
```

```
    /*
     * This method reads and checks inputs from the user. Arguments: Scanner, string
     * message and number value. Returns input from user to a variable. If the input
     * is invalid, an error message is shown.
     */
    private static double readValue(Scanner scanner, String message, double definition) {
        System.out.println(message);
        String input = scanner.nextLine();
        while (input != null) {
            Double value = null;

            if (input.equalsIgnoreCase("default")) {
                break;
            }

            if (input.trim().isEmpty() || (value = getDouble(input)) == null) {
                System.out.println(input + " is not a valid input. Try again!");
                System.out.println(message);
            }

            if (value != null) {
                return value;
            }

            if (scanner.hasNextLine()) {
                input = scanner.nextLine();
            } else {
                input = null;
            }

        }
        System.out.println("Using default value of " + definition);
        return definition;
    }

    /*
     * This method converts string input to double. Argument: String input from
     * user. Returns double.
     */
    private static Double getDouble(String input) {
        try {
            return Double.parseDouble(input);
        } catch (NumberFormatException ignored) {
            return null;
        }
    }

    /*
     * This method checks if the user input is empty. Arguments: scanner, string
     * message and default string. Returns string input or default input. It is used
     * to read file input from the user.
     */
    public static String readUserString(Scanner scanner, String message, String defaultValue) {
        // Prompt user
        System.out.print(message);
        // Save input
        String strInput = scanner.nextLine();
        if (strInput.trim().isEmpty()) {
            return defaultValue;
        }
        return strInput;
    }

    /*
     * This method checks if the user is choosing a valid choice. Arguments:
     * Scanner, String message and choices. Returns valid choice or keeps asking
     * user for new input.
     */
    public static String readChoice(Scanner scanner, String message, String... choices) {
        while (true) {
            System.out.print(message);
            String choice = scanner.nextLine().trim();
            int i = 0;
            while (i < choices.length) {
                if (choice.equalsIgnoreCase(choices[i])) {
                    return choice;
                }
                i = i + 1;
            }
            System.out.println("Invalid choice: " + choice);
            System.out.println("Valid choices: " + String.join(",", choices));
        }
    }

}
```

Bilag II (TestMedGrid.txt)

1.0

7

5

2

b

1

b

3

b

3

3

50