1 Introduction

Le but de ce projet est de produire un algorithme efficace pour résoudre le problème dessss

2 Solution proposé - Algorithme

Comme conseillé dans l'énoncé, on considere chaque état possible du parking comme un sommet dans un graphe, et chaque mouvement (déplacer une voiture de une position) un arc. Le problème devient alors très simple et il suffit de réaliser un BFS (Breadth First Search), qui visite les noeuds en largeur.

Plus spécifiquement,

Dans ce cas, puisque tous les arcs ont le même poids,

2.1 Parsing fichier input

Le fichier input passé comme argument au programme est "parsé" avec deux méthodes :

1. `parseFile()` : Utilise la classe `Scanner` et `FileReader` pour l'IO et se sert de la librairie `java.util.regex` pour établir un modèle de coordonnées et faire du "Pattern Matching".
2. `extractCar()` : Renvoie un objet de type `Car` avec les bonnes coordonnées.

2.2 Contraintes

— Toutes les voitures doivent être de taille 2.
— Les fichier input determine beaucoup des contraintes

2.3 Structures de données

3 Améliorations

Même si le paradigme OOP a été suivi, beaucoup de proprietés des objets restent publiques, pour faciliter l'accés aux autres classes, et ne pas gonfler les classes avec des accesseurs. Néanmoins, une meilleure encapsulation peut être achevée.

Un autre algorithme heuristique..

4 Code listing

```
import java.io.*;
import java.util.*;
import java.util.regex.*;

public class Escaper {
    public static void main(String[] args) {
        // Assuming no error in argument number
        System.out.println("\nInput file given: " + args[0]);
        Parking myPark = parseFile(args[0]);
        //myPark.printGrid();
        //System.out.println(myPark);
        bfs(myPark);
        //myPark.testing();
    }

    public static Parking parseFile(String file) {
        Scanner in = null;
        try {
            // Cheking if file is OK to read from
            in = new Scanner((new FileReader(file)));
        } catch (IOException e) {
            System.out.println("Wrong input file: " + e.getMessage());
        }
        // \\A in the beginning of the file. Interprets the rest as one big string.
        String text = in.useDelimiter("\\A").next();
        Pattern carPattern = Pattern.compile("\\Q[(\\E[0-4],[0-4]\\Q), (\\E[0-4],[0-4]\\Q)]\\E");
        Matcher matcher = carPattern.matcher(text);

        List<Car> carList = new ArrayList();
        boolean goal = true;
        int carCount = 0;
        while (matcher.find()) {
            // Goal car first. Should be set up like this in input.txt
            Car tempCar = extractCar(matcher.group(), goal, carCount);
            carList.add(tempCar);
            goal = false;
            ++carCount;
        }
        Parking myPark = new Parking(carList);
        return myPark;
    }

    public static Car extractCar(String ugly, boolean goal, int carCount)
    {
        String carId;
        int[] coords = new int[4];
        int index = 0;
        for (int i = 0; i < ugly.length(); i++){
            char c = ugly.charAt(i);
```

```
49              if (Character.isDigit(c)) {
                    coords[index] = Character.getNumericValue(c);
51                  index++;
                }
53          }
            if (carCount == 0) {
55              carId = "GG";
            } else {
57              carId = "c" + carCount;
            }
59
            Car car = new Car(coords[0], coords[1], coords[2], coords[3],
    carId, goal);
61          return car;
        }
63
        public static void bfs(Parking park) {
65          Queue<Parking> toCheck = new LinkedList();
            List<String[][]> visited = new ArrayList();
67
            // Check if goal
69          visited.add(park.parking);
            toCheck.add(park);
71
            escape:
73          while(toCheck.size() != 0) {
                Parking treating = toCheck.remove();
75              for (int i = 0; i < treating.moves.size(); i++) {
                    Move nextMove = treating.moves.get(i);
77                  Parking newParking = new Parking(nextMove);
                    if (isInList(visited, newParking.parking) == false) {
79                      System.out.println(visited.size());
                        if (checkSolved(newParking.parking)) {
81                          followPath(newParking);
                            System.out.println("solved");
83                          break escape;
                        }
85                      visited.add(newParking.parking);
                        toCheck.add(newParking);
87                  }
                }
89          }
        }
91
        public static boolean isInList(List<String[][]> list, String[][]
    candidate) {
93          // Comparing arrays, since the methods .equals() and
            // .contains() do not work well with arrays
95          for (String[][] array : list) {
                if (Arrays.deepEquals(array, candidate)) {
97                  return true;
                }
99          }
```

```
        return false;
    }

    public static boolean checkSolved(String[][] parking) {
        // Condition for the puzzle to be solved. Can be
        // anything... Testing the string in the exit case is simple
        return parking[Parking.exitX][Parking.exitY] == null ? false :
            parking[Parking.exitX][Parking.exitY].equals("GG");
    }

    public static void followPath(Parking solution) {
        // Recursive function that goes all the way up to the starting
        // point and then indicates the moves to do to reach the
        // solution.
        if (solution.comingFrom != null) {
            followPath(solution.comingFrom.predParking);
        }
        solution.printGrid();
    }

}
```

Escaper.java

```java
import java.util.*;


public class Parking {
    // Parking properties
    static int SIZE = 5;
    public static int exitX = 2;
    public static int exitY = 4;
    List<Car> carList = new ArrayList();
    public List<Move> moves = new ArrayList();
    public String[][] parking = new String[SIZE][SIZE];
    public Move comingFrom = null;

    public Parking (List<Car> carList) {
        // Default ctor
        this.carList = carList;
        updateGrid();
    }

    public Parking (Move move) {
        // Overloaded ctor if parking changes state
        this.comingFrom = move;
        this.carList = move.carList;
        Car toChange = carList.get(move.index);
        if (move.axis.equals("y")) {
            toChange.y1 += move.inc;
            toChange.y2 += move.inc;
        } else {
            toChange.x1 += move.inc;
            toChange.x2 += move.inc;
        }
        updateGrid();
    }

    public void addCar(Car car) {
        // Fill carList. No need if carList is public
        carList.add(car);
    }


    public String toString() {
        // Basic info to begin the program with.
        String rep = "Le␣parking␣a␣une␣dimension␣" + SIZE + "␣fois␣" +
   SIZE + "\n";
        rep = rep.concat("Il␣contient␣1␣Goal␣car␣et␣" + (carList.size() -
    1) + "␣autres␣voitures\n");
        for (int i = 0; i < carList.size(); i++) {
            Car car = carList.get(i);
            if (i == 0) {
                rep = rep.concat("La␣voiture␣Goal␣se␣trouve␣en␣position:␣
   ");
            } else {
                rep = rep.concat("La␣voiture␣" + i + "␣se␣trouve␣en␣
```

```
    position: ");
                }
52              rep = rep.concat(car.toString());
            }
54          return rep;
        }
56
        public void printGrid() {
58          // Sends out the grid.
            printGridPlus();
60          for (int i = 0; i < SIZE; i++) {
                System.out.print("|");
62              for (int j = 0; j < SIZE; j++) {
                    if (j != 0) {
64                      System.out.print(" ");
                    }
66                  if (parking[i][j] != null) {
                        System.out.print(" " + parking[i][j]);
68                  } else {
                        System.out.print("   ");
70                  }
                }
72              if (i != exitX) {
                    System.out.print("|");
74              }
                if (i != SIZE - 1) {
76                  printGridNextLine();
                }
78          }
            printGridPlus();
80      }

82      public void printGridPlus() {
            // Helper method
84          System.out.print("\n+");
            for (int i = 0; i < SIZE; i++) {
86              System.out.print("---+");
            }
88          System.out.println("");
        }
90

92      public void printGridNextLine() {
            // Helper method
94          System.out.print("\n+");
            for (int k = 0; k < SIZE; k++) {
96              System.out.print("   +");
            }
98          System.out.println("");
        }
100
        public List<Car> copyList() {
102         // Deep copy of carList
```

```
        List<Car> newList = new ArrayList();
        for (Car c : carList) {
            newList.add(new Car(c.x1, c.y1, c.x2, c.y2, c.carId, c.goal))
    ;
        }
        return newList;
    }

    public void updateGrid() {
        // Redraw every car in the grid and check for possible moves
        // where there are whitespaces
        for (int i = 0; i < carList.size(); i++) {
            Car car = carList.get(i);
            parking[car.x1][car.y1] = car.carId;
            parking[car.x2][car.y2] = car.carId;
        }
        for (int i = 0; i < carList.size(); i++) {
            Car car = carList.get(i);
            System.out.println(car);
            if (car.horizontal) {
                if (inboundsAndFree(car.x1, car.y1 - 1)) {
                    moves.add(new Move(copyList(), i, "y", -1, this));
                }
                if (inboundsAndFree(car.x2, car.y2 + 1)) {
                    moves.add(new Move(copyList(), i, "y", 1, this));
                }
            } else {
                if (inboundsAndFree(car.x1 - 1, car.y1)) {
                    moves.add(new Move(copyList(), i, "x", -1, this));
                }
                if (inboundsAndFree(car.x2 + 1, car.y2)) {
                    moves.add(new Move(copyList(), i, "x", 1, this));
                }
            }
        }
    }


    public boolean inboundsAndFree(int x, int y) {
        // Helper method to check if a case (x, y) is within the grid
        // and there is nothing on it.
        if ((x >= 0 && x < SIZE) &&
            (y >= 0 && y < SIZE)) {
            if (parking[x][y] == null) {
                return true;
            }
        }
        return false;
    }
}
```

Parking.java

```java
public class Car {
    public String carId;
    public int x1, y1, x2, y2;
    public boolean horizontal = false;
    public boolean goal = false;

    public Car (int x1, int y1, int x2, int y2, String carId, boolean goal) {
        this.goal = goal;
        this.carId = carId;

        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
        if (x1 == x2) {
            this.horizontal = true;
            if (y2 < y1) {
                this.y2 = y1;
                this.y1 = y2;
            }
        } else if (x2 < x1) {
            this.x2 = x1;
            this.x1 = x2;
        }
    }


    public boolean isGoal() {
        return goal;
    }

    public String toString() {
        // Coords
        return String.format("[(%d,%d), (%d,%d)]\n", this.x1, this.y1, this.x2, this.y2);
    }
}
```

Car.java

```java
import java.util.*;


public class Move {
    public List<Car> carList;
    public Parking predParking;
    public int index;
    public String axis;
    public int inc;

    public Move (List<Car> carList, int index, String axis, int inc,
   Parking predParking) {
        this.carList = carList;
        this.index = index;
        this.axis = axis;
        this.inc = inc;
        this.predParking = predParking;
    }
}
```

Move.java

```
Parking: 5 fois 5
+---+---+---+---+---+
|                   |
+   +   +   +   +   +
|                   |
+   +   +   +   +   +
|
+   +   +   +   +   +
|                   |
+   +   +   +   +   +
|                   |
+---+---+---+---+---+
Elements du Parking:
         voiture Goal: 1
         Autres voitures: 3
Emplacements:
         voiture Goal: [(2,0), (2,1)]
         voiture 1: [(0,1), (0,2)]
         voiture 2: [(1,2), (2,2)]
         voiture 3: [(2,3), (3,3)]
```

input.txt