|  |  |
| --- | --- |
| Project title | Paid parking lot |
| Author | Șandor Cristian Ionuț |
| Group | 30421 |

1. Task description

“Simulate partially the activity of a paid parking lot. The lot has a capacity of up to 50 cars.

You are to create a dashboard for the parking supervisor.

Cars come in through one entry which is provided with a barrier. They receive a ticket upon entry. Then the car is driven to an available lot. There is a queue of cars at the entry. Cars are admitted only if there is enough room for them. When a car attempts to leave the parking through the exit point, they are charged according to the time spent in the parking. There is a maximum time limit for parking, and charges vary depending on the time of day the car is parked. At any moment, the supervisor should know what cars are in the parking (by their license plate), how many available places are there, how much parking fees were collected so far during a day.

The simulation takes place in 1-minute steps.

You are supposed to simulate the activity the parking lot and provide a view of this.

All the relevant data (initial configuration, schedule, history data) should be stored in flat files.

The application should allow for saving the state and restoring it (serialization). A dashboard view should show the activity involved in this simulation. This should be a GUI which you should design.”

My approach of the problem consists in using 3 fundamental classes to manage the parking lot: CAR class, PARKING SPOT class and PARKING LOT class. Starting with the car, this class stores a plate number to identify the different cars. The parking spot has an ID to differentiate the spots and also stores the availability of the spot, the occupying time and a car. The parking lot, basically has an array of parking spots and general information as the time of the day and the gathered profit.

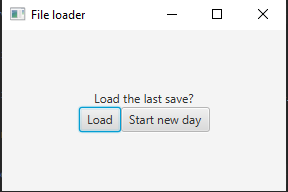
At the entrance every car receives a number of a spot which is available. The car occupies the spot and its spending time in the parking is monitored. Every car is charged according to the time spent and I chose the price to be 0.03 RON per minute. Between 12 o’clock and 15 o’clock the cars are charged with 0.05 RON per minute.

I also chose the time limit for every car in the parking to be 2 hours, after that time the car is automatically removed from the parking lot and it’s charged for that time.

If the parking lot is full the cars which are willing to enter will wait in a queue and every time a place is available the first car from the queue will occupy that place.

The simulation takes place in 1-minute steps so I chose to add a car every minute (so we can clearly see the functionality of the queue) and the cars which are leaving are randomly generated with a probability of 50% (when the parking is full). The parking lot is open from 07:00 to 18:00. When the closing time is reached, all the cars from the parking are removed and it’s shown the total profit from the day.

My implementation of the saving option is not complex. There is a “SAVE” button which, when pressed, the entire information from the parking lot is saved in a file. When the program starts it askes the user if he/she wants to load the saved data from the file. If the user chooses “LOAD” the program initializes the parking with the stored information, otherwise a new day starts.



The GUI is rather simple and easy to use and I designed it using java FX. It its structured in 4 parts. The top part shows information about the time and the gathered profit until the current time. In the right part we have the display of the parking lot state with all the parking lots. If the place is available it has a green color and is called empty. Otherwise it has a red color and the license plate number of the car is shown on the occupied spot. The left part of the window shows the status of the queue and all the cars which are waiting to enter the parking lot. In the bottom part there are several option buttons:

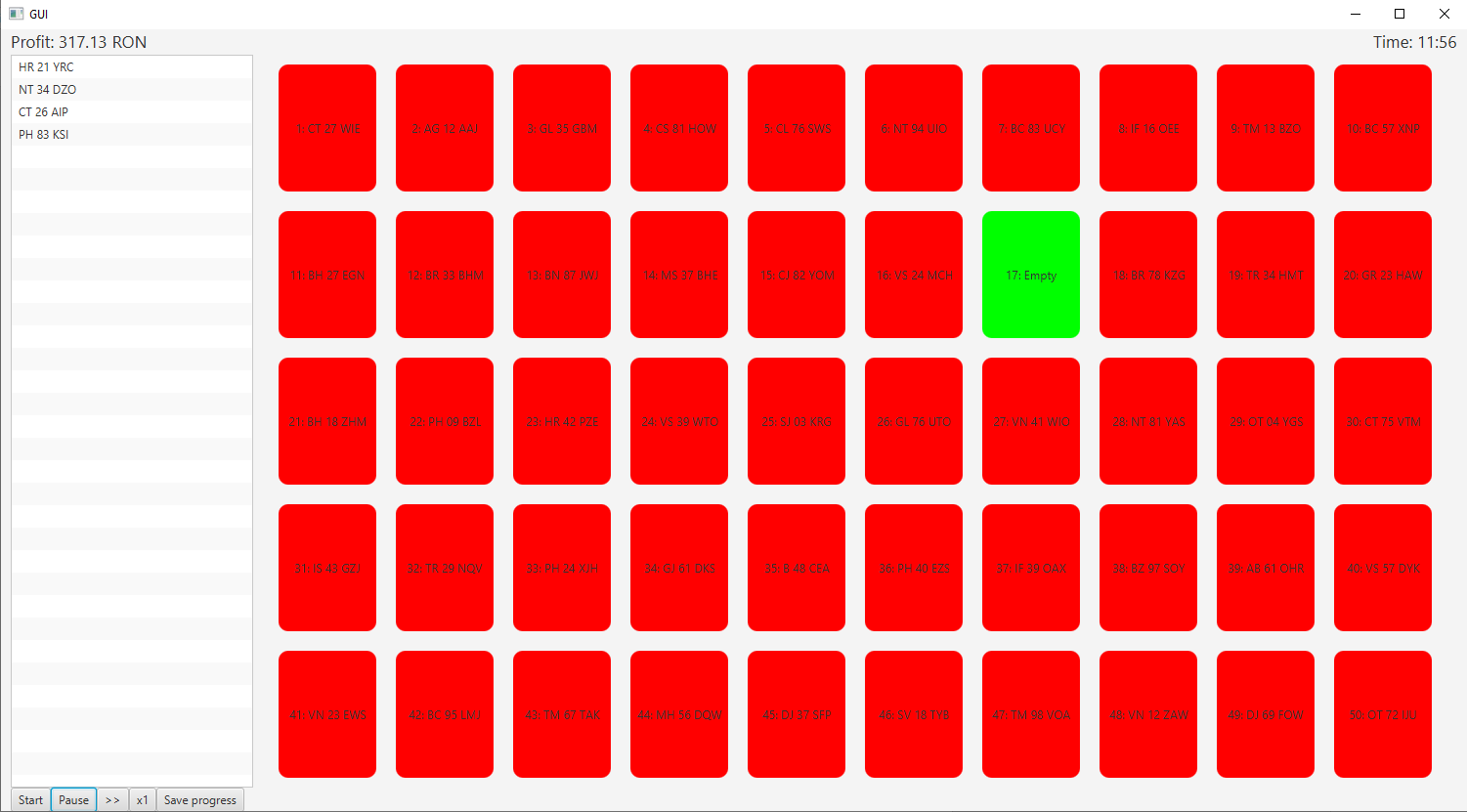
“START” – starts the simulation (used only after PAUSE);

“PAUSE” – pauses the simulation;

“>>” – x2 fast forward button, doubles the speed of the simulation (can be pressed multiple times);

“x1” – restores the simulation time;

“SAVE PROGRESS” – saves the state of the parking;



1. Class discovery

The project is structured in packages: Domain, Repository, Controller, View

View

Controller

Repository

Domain

Domain package:

Parking\_lot

Car

Parking\_spot

|  |  |
| --- | --- |
| PARKING SPOT | |
| * Assigns a car to a spot or frees a spot * Sends spot information to be saved | Car |

|  |  |
| --- | --- |
| PARKING LOT | |
| * Parks and unparks a car * Keeps track of available and occupied spots * Keeps track of time * Calculates profit * Empties the parking | Parking spot |

|  |  |
| --- | --- |
| REPOSITORY | |
| * Simulates step * Saves to file; Gets from file | Parking lot |

|  |  |
| --- | --- |
| CONTROLLER | |
| * Generates inputs for simulation * Simulates step depending on inputs and the state of the parking * Sends information to GUI * Converts time from minutes to hours and minutes | Repository |

|  |  |
| --- | --- |
| PARKING LOT WINDOW | |
| * Initializes, manages and updates the interface and its aspect | Controller |

|  |  |
| --- | --- |
| MAIN | |
| * Launches the program | Controller  ParkingLotWindow |

|  |
| --- |
| Car |
| -plate\_nr: String |
| +Car(plate\_nr: String)  +getPlate\_nr()  +toString() |

|  |
| --- |
| Parking\_spot |
| -counter: int  -spotID: int  -available: boolean  -initial\_time: int  -parked\_car: Car |
| +isAvailable()  +getTime()  +getSpotID()  +toString()  +Parking\_spot()  +park(car: Car, time: int)  +Parking\_spot(spotID: int, available: boolean, initial\_time: int, parked\_car: Car)  +empty\_park\_spot()  +toFile() |

|  |
| --- |
| Parking\_lot |
| -current\_time: int  -closing\_time: int  -profit: double  -available\_spots: ArrayList<Parking\_spot>  -occupied\_spots: ArrayList<Parking\_spot> |
| +Parking\_lot()  +Parking\_lot(parkingArray: ArrayList<Parking\_spot>, time: int, profit: double)  +parkCar(car: Car)  +exitCar(spotID: int)  +calculateProfit(p: Parking\_spot)  +getCurrent\_time()  +getProfit()  +getAvailable\_spots()  +getOccupied\_spots()  +getClosing\_time()  +checkTimeLimit()  +isFull()  +incrementTime()  +closeParkingLot() |

|  |
| --- |
| Repository |
| -parkingLot: Parking\_lot  -historyData: String |
| +Repository(p: Parking\_lot)  +simulateStep(enteringCar: Car, exitID: int)  +getClosingTime()  +getTime()  +getProfit()  +getOccupied\_spots()  +getAvailable\_spots()  +isFull()  +saveToFile()  +getFromFile() |

|  |
| --- |
| Controller |
| -repos: Repository  -queue: Queue<Car>  -county: ArrayList<String>  -characters: String |
| +Controller(repos: Repository)  +generateRandExit()  +generatePlateNr()  +generateCar()  +simulateNextStep()  +getTotalSpots()  +getTime()  +getProfit()  +getQueue()  +saveToFile() |

|  |
| --- |
| ParkingLotWindow |
| -queueView: ListView<String>  -borderPane: BorderPane  -fastForwardButon: Button  -playButton: Button  -pauseButton: Button  -normalSpeedButton: Button  -saveButton: Button  -ctrl: Controller  -playSpeed: int  -timeDelay: Timeline  -exitButton: Button  -parent: Stage  -parkingView: TilePane  -profitLaber: Label  -clockLabel: Label |
| +initialize(ctrl: Controller, parent: Stage)  +getExitButton()  +setSpeed(newSpeed: int)  +update()  +startSim()  +endWindow(profit: String) |

|  |
| --- |
| Main |
| -window: Stage  -parkingScene: Scene  -parkingController: ParkingLotWindow  -ctrl: Controller |
| +start(primaryStage: Stage)  +main(args: String[]) |