Matthew Stockert & Dominic Retli
April 5, 2018
CIS 450-001
Professor Wang

P3

Table of Contents

Files Submitted

P3_450.pdf – This report
tc.java – Main class
Car.java – Car object and thread for car
Point.java – Each Point on the intersection is an instance of this class
Points.java – A collection of Point.java
Directions.java – Each car will create an instance this class

Implementation

tc.java

- This is our main class. This class will create the Point objects as well as the separate threads for cars.
    - Main()
        - This method calls a method that creates Points representing the intersection.
        - This method calls a method that creates the threads representing each car.

Car.java

- This class represents a car, as well as a thread, combined in one.
    - Car()
        - This is the constructor. It sets the Directions object for the car (this contains the original direction and the target direction).
        - This method will also sleep for the appropriate time given in the assignment details in order to create the different arrival times of each car.
    - run()
        - this method overrides the run() method of the Thread class. Car extends the thread class. In this method, the three methods described in the assignment are called: (arrive, cross, and exit)
    - ArriveIntersection()
        - This method is called to simulate the arrival of the car at the intersection.
        - The needed Point objects (semaphores) needed to complete the desired turn are obtained. From here, the first need point (representing a stop sign) is acquired. After that, another semaphore called “isNext” is acquired. This isNext semaphore is used to ensure that cars turn in a first come first server pattern, as described in the assignment details.
        - The remaining needed points are looped through and acquired.
    - CrossIntersection()
        - This method is called to simulate the crossing of the car at the intersection.
        - First, the appropriate stop sign semaphore is released.

        - Next, the thread will sleep for the appropriate time depending on the turn type (left, right, or straight turns have different times given in the assignment details).
    - ExitIntersection()
        - This method is used to release all of the semaphores, one by one, that were acquired by the Car as the turn is being complete. We do this one by one so that another car, going the same direction, does not have to wait until the car in front of it has completed its entire turn (as requested by Prof. Wang).
    - Print()
        - This method prints the details of each car action (arrival, cross, exit) with the appropriate time, car id, original direction, target direction, and action performed.

Point.java

- This class is essentially a wrapper for the java Semaphore class
    - The Point class contains a Semaphore as a data member.
    - Each point was assigned an index1 and index2, in order to match with our diagram attached to the bottom of this report (See image of intersection for more details).
    - acquireLock()
        - This method will acquire the semaphore and stores the Direction object and cid of the Car obtaining the lock.
    - releaseLock()
        - This method will release the semaphore of the Point class.
    - noPermit()
        - this method is used to verify whether the semaphore in the Point class is locked or not. If there are no permits, then the noPermit() method will return true, else return false.
    - lockedBy()
        - this method returns the cid stored in the Point class
    - numPermits()
        - this method returns the number of permits of the Semaphore class.

Points.java

- This class contains several ‘global’ static methods and data structures
    - ArrayList<ArrayList<Point>> points
        - This 2d array contains all of the Point objects (semaphores) in the arrangement of an intersection. Each Point object represents a point in the intersection (see image of intersection below for more details).
    - Point isNext

        - This Point object (semaphore) allows a Car to claim the next spot, or right-of-way. This ensures a first come first serve pattern just like a real intersection (as described in the assignment).
    - startTime
        - startTime is a LocalTime variable used to calculate the times of arrival, cross, and exit in the output seen below.
    - createPoints()
        - This method is used to populate the ArrayList<ArrayList<Point>> points array (intersection) with points.

Directions.java

- The Directions class essentially represents the turn that will take place
    - Dir_original
        - This char represents the original direction a car was heading
    - Dir_target
        - This char represents the direction the car wants to head.
    - turnType()
        - this method returns a 1,2, or 3 depending on the turn type (Right Left or Straight). This is later used to sleep the thread for the appropriate time during the CrossIntersecіont() method of Car.java.
    - pointsNeeded()
        - this method returns an array of Point objects (Semaphore), based on the turn that is needed to be performed. For example, if a Car is started heading north and wants to continue heading north, they will need to acquire the following Points/semaphores: 0-0, 1-0, 2-0, 2-3, and 4-2. The method looks at the Direction's dir_target and dir_original to figure this out.
    - getDir_original()
        - returns dir_original
    - setDir_original()
        - sets dir_original
    - getDir_target()
        - returns dir_target
    - setDir_target()
        - sets dir_target

Report

Screenshots

Matthew Stockert and Dominic Retli

Report

To implement the stop sign problem we had 21 semaphores. Each semaphore corresponds to a point on the map. We chose to store these semaphores in a 2-dimensional array of points. The car will first acquire the stop sign semaphore (0-X) where X is the stop sign they are at. Once they acquire the stop sign they have to acquire the isNext semaphore to let all other cars no they are top priority. Once they acquire both of these semaphores they will acquire the rest of the points they will need and once they have all points they need they can make their turn. A sample diagram of points a car needs to acquire are down below, showing a right turn, left turn, and a car going straight. The red points are the points they will need to acquire. The map can be rotated (without rotating the numbers) to show the rest of the turns. The point maps allow two left turns from opposite directions (for example, ->N ->W and ->S ->E). If a car coming from the same direction as in the middle of crossing will obtain the locks, allowing that car to turn as well.

Diagrams

Figure 1 – Right Turn

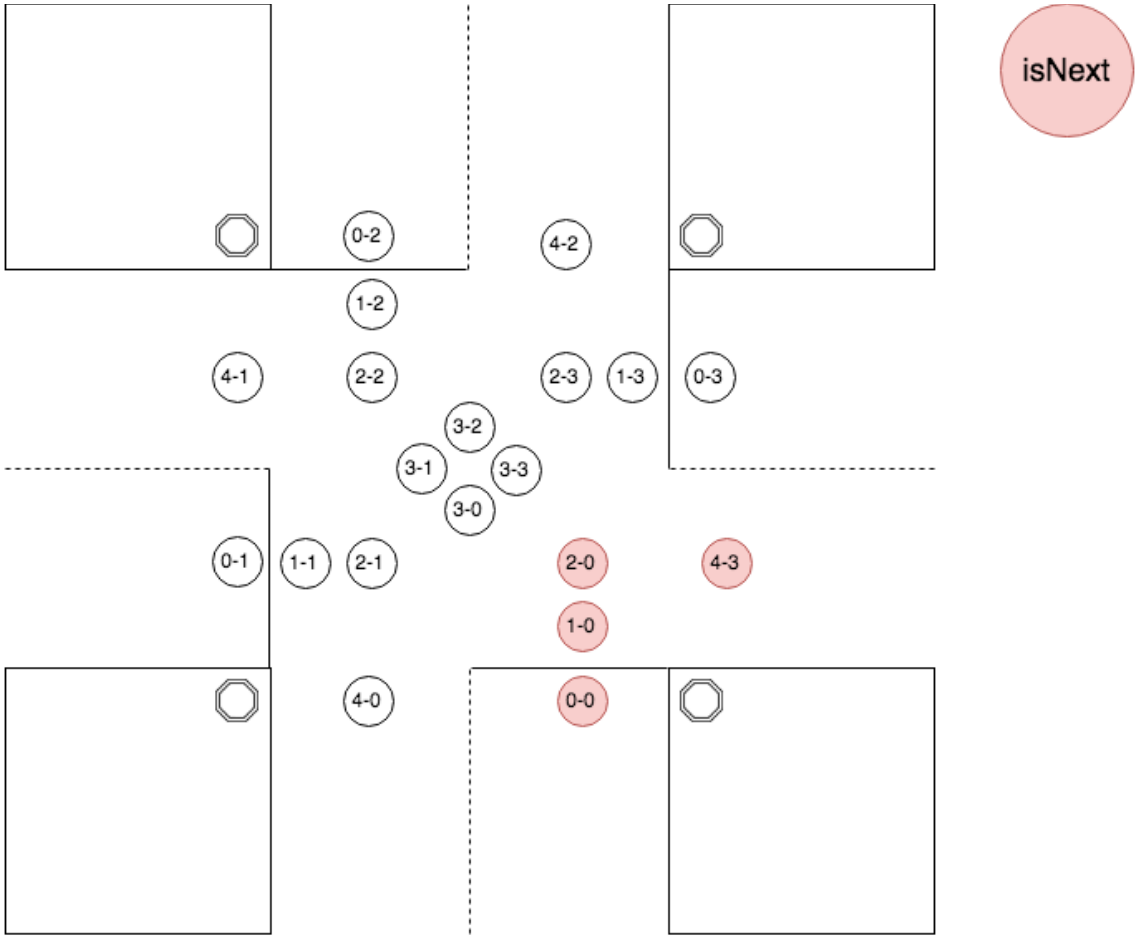


Figure 2 – Straight

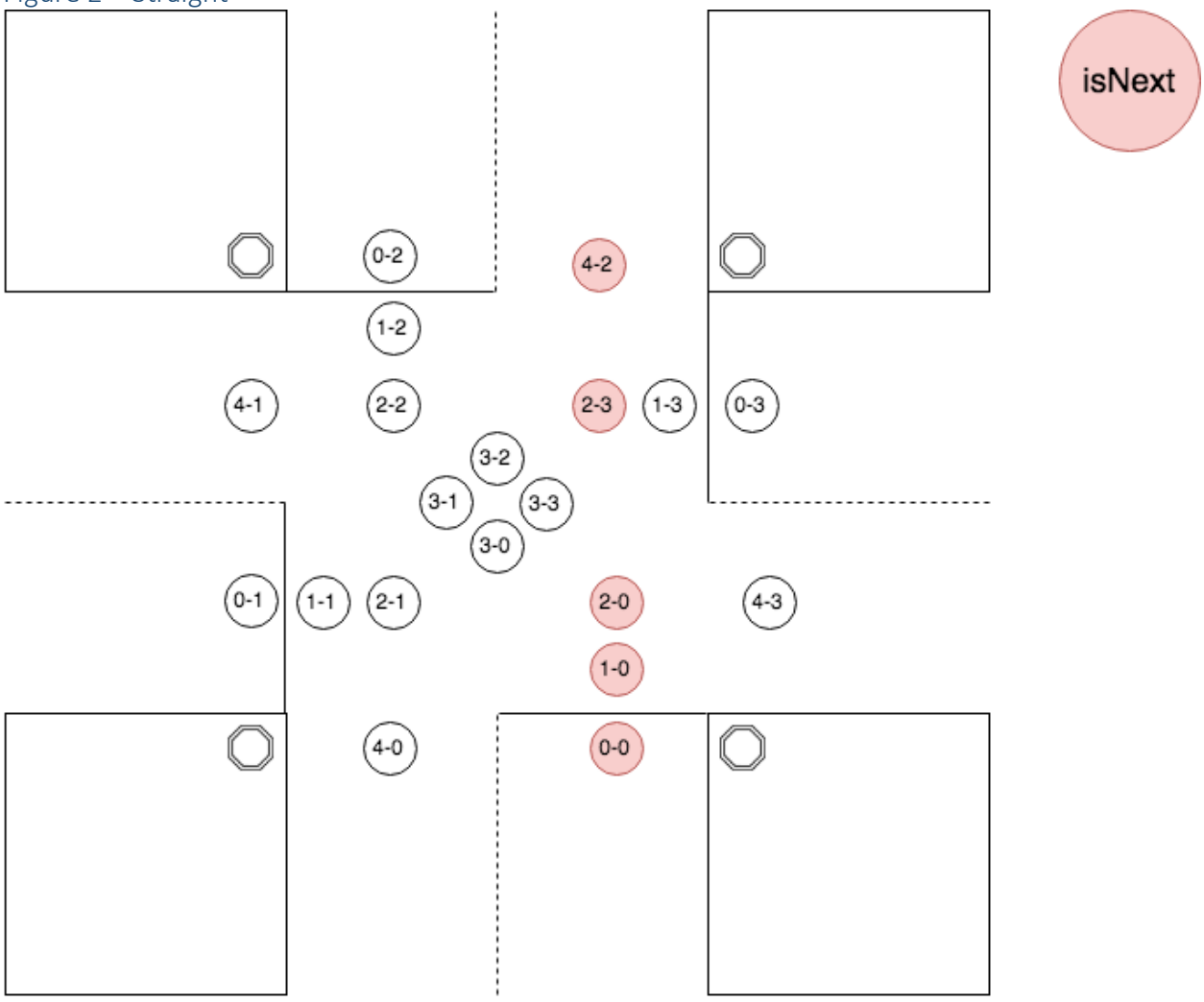
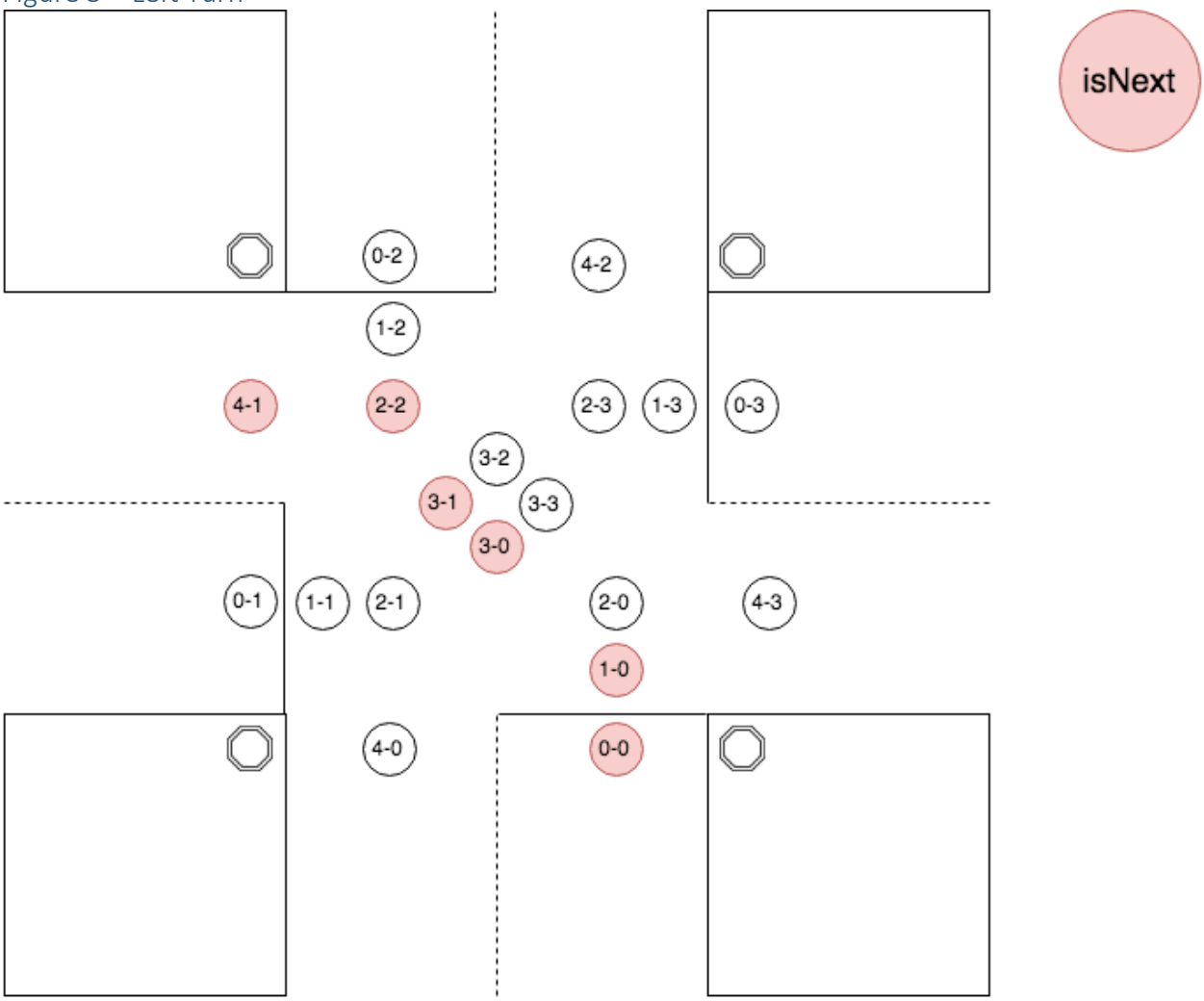


Figure 3 – Left Turn



Contribution

Dominic Retli – Implementation of Document, Point Object of code, Logic of stop signs
Matthew Stockert – Report of Document, Car Object of code, Logic of stop signs