Lab1 - trainspotting documentation

Sara Borg, Wanda Wannelöf

January 2023

1 Introduction

In order to learn about parallel programming and the different tools which are used for it, we were given the task to make two trains go between two stations simultaneously, without derailing or crash into each other.

Our general solution was that the trains always chooses the straight path at each junction, which meant that after a train had passed a switch, the switch always changed back to direct to the straight path.

Our solution is illustrated in Figure 1, which illustrates the sensor placements and the critical sections we've chosen.

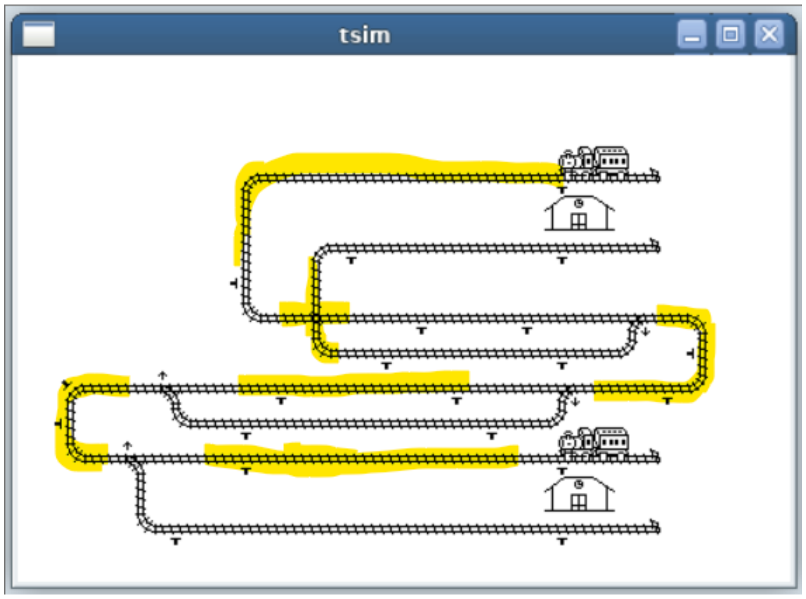

Figure 1: Picture of our map with the placements of the sensors. The yellow area marks the critical sections

2 Placement of sensors

When considering the placement of the sections, we figured that it was necessary to know when a train enters and exits any part of the track. This led to us placing a sensor after and before each switch as well as the crossing. Then, in order for the trains to stop and wait at each station, we'd need to know when they arrive there. This led to a sensor at each station. The result was 20 sensors with the following sensor names:

```
enum Sensors {
    CROSSING_W, CROSSING_S, CROSSING_E, CROSSING_N,
    EAST_E, EAST_S, EAST_W,
    MID_E, MID_S, MID_W,
    WEST_W, WEST_E, WEST_S,
    SOUTH_W, SOUTH_E, SOUTH_S,
    NORTH_STATION_N, NORTH_STATION_S,
    SOUTH_STATION_N, SOUTH_STATION_S
}
```

The precise placement of each sensor was determined by the braking distance of when the trains have the decided maximum speed. After testing, we found that placing the sensors three spaces apart from the switch/crossing so that the train always manages to stop. Same goes for the placement at the stations, the train always comes to a stop so that it never derails off the track.

3 Choice of critical sections

In order to figure out where our critical sections are, we first had to decide which tracks are the default route to take. We figured the best choice for the default route would be where the trains would go straight ahead as much as possible, which to us also seems to be the shortest route.

So what parts became critical? Every part of the default route where there aren't any optional tracks to deviate to in case there would enter another train on the same track. This resulted in the following six sections:

- North track by the North Station, `NORTH_STATION_N`
- The crossing right below North Station, `CROSSING`
- East part of the track, `EAST_TRACK`
- Middle part of the track, `MID_TRACK`
- West part of the track, `WEST_TRACK`
- North part by the South Station, `SOUTH_STATION_N`

In the code, each critical section is implemented as a semaphore, where each semaphore can only be occupied by at most one train:

```
enum Semaphores {
    N_STATION, CROSSING, EAST_TRACK,
    WEST_TRACK, MID_TRACK, S_STATION
}
```

4 Maximum train speed

We started by using the max speed 15 and decide the placement of the sensors after that. But after the solution worked with that speed, we tested increasing the speed. The braking distance obviously increased which meant that in order to increase the maximum speed we also needed to modify the placement of the sensors. We decided to shift each sensor one position away from the critical areas' entrances (the switches and the crossing).

The maximum speed we then decided on was 22. With this speed, it always manages to come to a full stop at the stations and switches/crossing.

5 Testing

We tested the maximum speed by trying with different simulation speeds. Primarily with the standard sim speed and sim speed = 1 (the fastest simulation speed). We tested with several combinations of speed to ensure that our solution works.