# ALP IV: Übung 4

Tutor: Christoph Beuck

Lisa Dohrmann (4130066), Adrian Neumann (4140810), Naja v. Schumde (4127652)

27. Mai 2008

## Aufgabe 1

In der Klasse `BlockSupport` legen wir unsere Methoden zum gegenseitigen Ausschluss fest. Zwei LKWs dürfen nicht gleichzeitig das selbe Road-Objekt belegen. Daher merken wir uns für jeden Straßenabschnitt den LKW-Thread, der sich gerade dort befindet. Will ein zweiter Thread ebenfalls diesen Abschnitt blockieren, wird er per `wait()` zur Kaffeepause gezwungen. Verlässt nun der erste den Abschnitt wieder, teilt er das einem zufällig gewählten und auf diesen Abschnitt wartenden Kollegen mit.

```
1  public class BlockSupport
2  {
3     private Thread current;
4
5    /**
6     * Liefert den Thread, der gerade das Objekt belegt hat, oder null.
7     * Achtung, dies sollte nur zu Diagnosezwecken verwendet werden!
8     * @return den belegenden Thread oder null
9     */
10   public synchronized Thread blocking() {
11       return current;
12   }
13
14   /**
15    * Belegt das Objekt.
16    * Ist das Objekt bereits belegt, kehrt diese Methode erst zurück,
17    * wenn es wieder freigegeben wird.
18    * @exception InterruptedException  ein wartender Thread wurde
          unterbrochen
19    */
20   public synchronized void block() throws InterruptedException {
21       while (current!=null) { //another thread is blocking
22          System.out.println("Can't move, too much traffic...");
23          wait();
24       }
25       current = Thread.currentThread();
26   }
27
28    /**
29    * Gibt das Objekt wieder frei.
30    * Warten andere Threads auf die Freigabe, so wird nichtdeterministisch
31    * einer davon ausgewählt und belegt als nächster das Objekt.
32    */
33    public synchronized void unblock() {
34       current = null;
35     notify();
36   }
37 }
```

Ein LKW, der sich über die RoadMap bewegt, muss also, um einen Crash zu verhindern, jeden zu befahrenden Abschnitt blockieren und nach Verlassen wieder freigeben. Das passiert in seiner `drive()` Methode.

```
void drive() {
   Random rnd = new Random();

   // try ahead first
   int steer = Road.AHEAD;
   if (myRoad.getExit(Road.nextDirection(myDirection, steer))== null) {
      // then left or right at random
      if(rnd.nextInt(2)==0) steer = Road.LEFT;
      else steer = Road.RIGHT;
   }
   if (myRoad.getExit(Road.nextDirection(myDirection, steer))== null) {
      // still no luck, try now the opposite
      if(steer == Road.LEFT) steer = Road.RIGHT;
      else steer = Road.LEFT;
   }
   if (myRoad.getExit(Road.nextDirection(myDirection, steer))== null) {
      throw new RuntimeException("Trapped!!"); // that's bad
   }
   myDirection = Road.nextDirection(myDirection,steer);
   Road nextR = myRoad.getExit(myDirection);

   // before leaving the old road, try to block the new one
   try { nextR.block(); }
   catch(InterruptedException e) {
      e.printStackTrace();
   }
   // ok, now move it
   myRoad.setTraffic(null);
   myRoad.unblock();
   myRoad = nextR;
   // clear the goal
   if(arrived()) {
      myRoad.unblock();
      return;
   }
   myRoad.setTraffic(this);
}
```

Ein LKW Objekt implementiert die Interfaces Truck und Runnable. Es kann daher als neuer Thread gestartet werden. Der LKW fährt dann solange durch die Gegend, bis er sein Ziel erreicht hat.

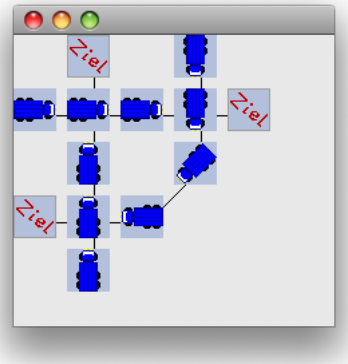
```
public void run() {
   for(; !arrived(); drive()) {
      map.roadChanged(); // force an update
      try { Thread.sleep(500); }
      catch(InterruptedException ex) {
         ex.printStackTrace();
      }
   }
   System.out.println("The␣truck␣'"+Thread.currentThread().getName()
      +"'␣arrived!");
}
```

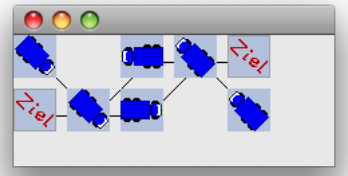
Durch Parameter kann das Szenario und die Anzahl der LKWs festgelegt werden. In einer for-Schleife wird dann die entsprechende Anzahl an LKW-Threads erstellt. Wurde beim Start kein Parameter für die Anzahl angegeben, wird zumindest einer gestartet. Hier der entsprechende Ausschnitt aus der Main-Klasse.

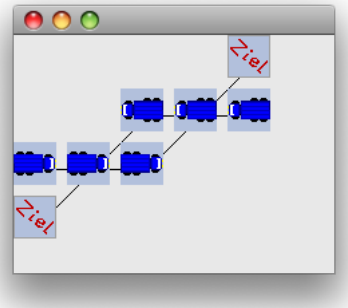
```
public static void main(String[] args) {
   //[...]
   int anz = 1; //at least one truck
   if (args.length >= 2) {
      try{ anz = Integer.parseInt(args[1]); }
      catch(Exception e) { usage(); }
   }

```

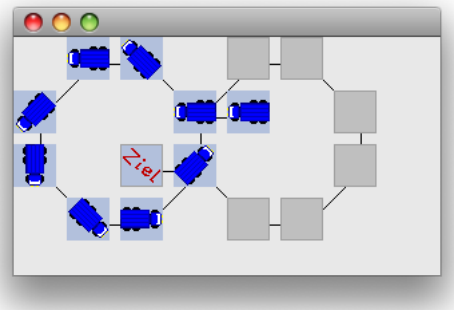
```
9       // create and place anz trucks, default is 1
10      for (int i=0; i<anz; i++) {
11          // find a free start road
12          do {
13              startR = roads.getStart(rnd.nextInt(roads.getStarts()));
14          } while(startR.getTraffic()!= null);
15
16          // find a proper start direction
17          int dir = Road.NORTH;
18          while(startR.getExit(dir) == null) dir++;
19          new Thread(new LKW(startR,dir,roads)).start();
20          roads.roadChanged();
21      }
22  }
```

Wenn man zu viele Trucks startet (zwischen 5 und 15 Trucks ca.), kommt es zu folgenden Bildern, wo sich nix mehr tut. Für viel Betrieb sind die Strecken also nicht ausgelegt.




Wenn man bei der Acht, den Ziel- und Startpunkt vertauscht, ändert dies nichts an der Situation, da es dann einfach gespiegelt ist - mit 8 Trucks ist hier in jedem Fall Schluss.
Bei InselA und InselB müsste es eigentlich funktionieren, da man so erreicht, dass alle Trucks in die selbe Richtung wollen, also immer jemand sein Feld freimachen kann, so dass die anderen nachrücken können.
Wenn man im Kreuzungs-Szenario Start und Ziel einer beliebigen Strecke vertauscht, sollte es auch klappen. Durch den Tausch gibt es dann immer eine Ecke an der zwei Zielpunkte liegen. Dadurch ist gesichert, dass es dort zu keinem Stau kommen kann und ein dort stehender Truck immer ins Ziel kommt. Das hat zur Folge, dass sich irgendwann auch an den anderen Ecken der Stau auflöst.