Opgave 14: Threadcommunicatie

– Practicum OO, lente 2015 –

Leerdoelen

Na het maken van deze opgave kun je:

- threads synchroniseren met behulp van `wait` en `notify`;
- kritieke secties maken voor threads.

Route 66

In deze opgave gaan we threads laten communiceren. Iedere thread bestuurt een auto en voorkomt aanrijdingen met andere auto's. Het totale verkeer wordt getoond in een grafische simulatie. In die grafische weergave is het goed te zien als er iet misgaat met de synchronisatie en wat er mis gaat. Er zijn twee views op het verkeer. In de roadView wordt het verkeer grafisch weergegeven. In de tableView staat per auto de positie als getal gegeven.

Het gegeven programma heeft een keyboard listener. De toets `s` zal de auto's laten stoppen, de toets `q` zal het programma stoppen. Met < en > kan de wachttijd tussen de stapjes kleiner respectievelijk groter gemaakt worden. Iedere andere toets zet de auto's weer in beweging als ze gestopt zijn.

Als het attribuut `DIRECTIONS` in `Model` de waarde 2 heeft is er slechts een weg met verkeer in twee richtingen. Als dit attribuut de waarde 4 heeft is er een kruising met verkeer in vier richtingen. Alle auto's zullen op de kruising rechtdoor rijden.

1 Threads

Zo als je ziet is de gegeven implementatie van Route66 erg gevaarlijk voor het verkeer. Alle auto's rijden op hun eigen snelheid zonder naar het overige verkeer te kijken. In de `Controller` doen de auto's steeds allemaal een stapje en dan wordt er met `pause` een pseudorandom tijdje gewacht.

Om het probleem van kop-staart botsingen op te lossen moet iedere auto een eigen bestuurder, een object uit de klasse `Driver`, krijgen. De drivers dienen aanrijdingen tussen de auto's te voorkomen. In dit onderdeel van deze opgave doen de drivers dit door de auto alleen verder te laten rijden als de voorganger voldoende ver weg is. Je zult hiervoor misschien methoden moeten toevoegen aan `Car`.

Ieder van deze drivers moet in een eigen thread gaan draaien. Je hoeft de threads hiervoor niet te synchroniseren, het is voldoende om de positie van de voorganger (de auto voor deze auto) te inspecteren. Laat de random stapgrootte en wachttijd bestaan! Je oplossing moet kijken of er na de verplaatsing ten gevolge van de `step` nog voldoende afstand tot de voorganger is. Als er voldoende afstand is laat de driver haar auto een stap doen. Anders gaat de driver korte pseudorandom tijd slapen en kijkt vervolgens of er nu wel een stap gedaan kan worden. Omdat de auto's elkaar nooit inhalen, hoeft je als driver altijd maar één voorganger te bekijken.

2 Wachten op voorganger

In bovenstaande oplossing kijken de drivers waarschijnlijk onnodig vaak of hun auto een stapje naar voren kan. Zo lang de auto voor deze auto niet bewegen heeft is er zeker geen stap mogelijk. Verbeter de driver zodat die gaan wachten totdat de voorganger een stapje heeft gedaan. Als er voldoende ruimte is zal de auto een stapje naar voren gaan, anders gaat de driver weer slapen.

De drivers auto's moeten via het toetsenbord nog steeds gestopt kunnen worden met (met een `s`), en door een ander toets weer in beweging gezet kunnen worden. Ook de snelheid moet geregeld kunnen worden met < en >. De enkele stap mag je laten vervallen als dat lastig is.

3 Blikschade

In de rest van de opgave laten we Route 66 een andere interstate kruisen door de constante `DIRECTIONS` in `Model` op 4 te zetten. Alle verkeer gaat rechtdoor over de kruising, maar moet natuurlijk wel blikschade door aanrijdingen gaan voorkomen.

Natuurlijk moeten de drivers nu zodanig aangepast worden dat botsingen op de kruising voorkomen worden. Om aanrijdingen op de kruising te voorkomen dien je een `Regelaar` aan je programma toe te voegen, of de bestaande `Controller` zo aan te passen dat hij deze taak vervult. Deze regelaar werkt min of meer als een stoplicht. De regelaar heeft minstens twee toestanden. In de eerste toestand mogen alleen auto's over route 66 in beide richtingen (oost-west en west-oost) over de kruising rijden. In de tweede toestand is dat alle verkeer over de kruisende interstate.

De regelaar die je moet maken is verkeersafhankelijk. Als er veel verkeer op een weg is zal dat verkeer langer toestemming krijgen om over de kruising te rijden. Zodra er geen verkeersaanbod meer is, krijgt het verkeer dat op de andere weg staat te wachten zo snel mogelijk de beurt. Om looks en starvation te voorkomen mogen er maar een beperkt aantal auto's in een bepaalde richting over de kruising rijden zodra er verkeer in de kruisende richting staat te wachten. De kruising is dus *niet* een kritieke sectie waar maar op ieder moment hoogstens één auto is. Meer verkeer op de kruising is prima, zo lang het maar geen kruisend verkeer is.

Zodra een driver de kruising wil gaan oprijden voert deze de methode `cross` van de regelaar uit. De driver geeft hierbij aan in welke richting hij over de kruising wil rijden. Het type van deze methode is dus iets als **void** `cross(Direction d)` Het uitvoeren van deze methode duurt zolang als nodig is om veilig verder te kunnen rijden. Als de driver de kruising verlaat meldt hij dit via het uitvoeren van de methode `leave` van de regelaar.

Inleveren

Voor zondag 31 mei, 11:00 uur, via Blackboard. Het is voldoende om alleen de laatste versie van de klassen in te leveren.