Werkoverleg Algoritmes, 21 oktober, *12:30-13:30*,
Auditorium 10, week 8,
TU/e Eindhoven, Faculteit Wiskunde en Informatica

Aanwezig: Alwin Penterman, Rick Hilkens (notulist).
Afwezig: geen van werkgroep Algoritmes.

Globale opbouw:
Hieronder volgen de hoofdlijnen van aantal belangrijke algoritmes aangaande Etaris:
Noot: onderstaande code is geen C# code, aangezien een aantal details weggelaten zijn.

```
class Piece { LinkedList Blocks; … }
class Block { Piece Parent; … }
LinkedList<Piece> Pieces;
```

Ieder blok (een `Block`) in Etaris behoort altijd bij een speelstuk (een `Piece`). Blokken en speelstukken hebben elkaars referentie. De variabele `Parent` functioneert als een soort ID.

```
Block[,] Grid;
```

De basis voor het spel is de tabel `Grid`. Ieder element `[i,j]` komt overeen met kolom `i` en rij `j` van het speelveld. Ieder element is een blok van een speelstuk (of `null`). Wanneer een blok verplaatst wordt, dan wordt het blok in `Grid` verplaatst.

Plaatsbepaling blok:
Plaats bepaling van een blok kan op diverse manieren geschieden:

```
class Block { Piece Parent; int i; int j; }
```
Waarbij `i` en `j` verwijzen naar een positie in `Grid`.

```
class Block { Piece Parent; Block leftNeighbour; Block bottomNeighbour; … }
```
Verwijzen naar de buren kan, maar werken met `i` en `j` is waarschijnlijk gemakkelijker.

Speelstuk opsplitsen:
Als een rij gewist wordt, is het mogelijk dat een speelstuk in meerdere delen wordt gebroken (bijvoorbeeld bij een U vorm, als de onderste rij gewist wordt). In dat geval verdwijnt het speelstuk en komen er twee of meerdere nieuwe voor in de plaats. Er zijn twee algoritmes mogelijk, vlak na de splitsing om de nieuwe speelstukken te construeren:

Algoritme 1:

1. Verwijder de gewiste blokken uit `Blocks` van het oorspronkelijke `Piece` object.
2. Neem een blok uit het speelstuk (na stap 1) en vind recursief aanliggende speelstukken welke behoren bij dit blok. Maak hier een nieuw speelstuk van.
3. Wanneer niets meer gevonden wordt, dan wordt nogmaals stap 2 toegepast op een blok van het speelstuk dat nog niet verwerkt is door stap 2 (of een andere stap 3).
4. Herhaal stap 2 en 3 totdat alle blokken waarnaar het speelstuk na stap 1 refereerde verwerkt zijn.

Algoritme 2:

1. Verwijder de gewiste blokken uit `Blocks` van het oorspronkelijke `Piece` object.
2. Neem een niet verwerkt blok uit `Blocks` van het speelstuk, voeg dit blok toe aan een nieuw speelstuk. Voeg alle aangrenzende blokken ook hieraan toe door alle blokken in de lijst `Blocks` te controleren. Controleer opnieuw op buren voor dit nieuw toegevoegde blok en voeg eventuele nieuwe buren ook toe aan het speelstuk (buren moeten altijd uit `Blocks` komen).
3. Wanneer geen nieuwe buren meer gevonden worden, kan stap 2 herhaald worden op een nog niet verwerkt blok.

Algoritme 1 is theoretisch sneller voor grote aantallen blokken. Wellicht dat het tweede algoritme sneller is voor kleine aantallen. Algoritme 2 is gemakkelijker te implementeren.

Speelstukken laten vallen:
Wanneer een rij verwijderd is en de speelstukken indien nodig gesplitst zijn, is het mogelijk dat er speelstukken zijn die kunnen vallen.

Algoritme:

1. Neem het eerste element uit `Pieces` (een verzameling van alle speelstukken).
2. Controleer of dit element kan vallen, zo ja, laat het maximaal vallen.
3. Loop de verzameling af en herhaal stap 2 voor ieder element.
4. Als gevolg van eerdere stappen 2, is het mogelijk dat anderen blokken kunnen vallen. Herhaal het hele algoritme totdat de situatie stabiel is, dat wil zeggen, geen enkel speelstuk kan meer vallen.

Dit algoritme is simpel en snel voor kleine aantallen speelstukken (zoals bij Etaris). Bij duizenden of miljoenen speelstukken zou een geavanceerder algoritme wenselijk zijn, dit is niet van toepassing.