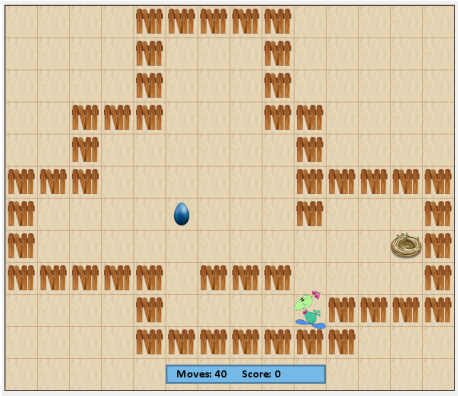
Opdracht 8 Sokoban

– Algoritmisch Denken en Gestructureerd Programmeren in Greenfoot –

1 Inleiding

In deze opdracht ga je in Greenfoot een spelletje schrijven dat bekend staat onder de naam *Sokoban*. Onze variant daarop heet 'Mimi de eitjesverzamelaar'. Het oorspronkelijke spel speelt zich af in een magazijn. De speler is een magazijnwerker die als taak heeft kratten naar een juiste plek te schuiven.



Maar dit spel speelt zich af in `Madagaskar`, de wereld waarin Mimi leeft. Mimi moet de eieren in de nestjes schuiven. Daarbij gelden de volgende regels:

- Mimi kan niet op een ei zitten of eroverheen stappen;
- Mimi kan een ei vooruit duwen; aan een ei trekken kan ze niet.
- Mimi kan maar één ei tegelijk verplaatsen. Dus als er twee eieren achter elkaar liggen dan kan Mimi die niet gelijktijdig verschuiven.
- Er kunnen geen twee eieren op dezelfde plek liggen (dus ook niet in een nestje).
- Nestjes kunnen maar één ei bevatten. Wel kan een een ei wat in een nest zit er weer uit geduwd worden.
- Noch Mimi, noch een ei, kan door of over een hek heen;
- Mimi kan over nestjes heen lopen (ze doet dat natuurlijk heel voorzichtig zodat ze ze niet kapot trapt).
- Eieren kunnen niet door hekjes heen geduwd worden.
- Er zijn evenveel nestjes als eieren en het spel is afgelopen zodra elk nestje gevuld is.

2 Leerdoelen

Na het voltooien van deze opdracht kun je:

- gebruikersinteractie in de code afhandelen;
- Geneste **`if`** .. `then` .. **`else`** statements toepassen;
- de kennis uit de vorige opdrachten toepassen om zelf een spel te implementeren.

3 Instructies

Voor deze opdracht heb je scenario **'MadagaskarSB'** nodig.

4 Uitleg

We beginnen met een uitleg van de een bepaald gebruik van **`if`** `then` **`else`** statements, deze keer in de context van Sokoban. We voegen nu namelijk gebruikersinteractie toe. Deze keer leggen we dus uit wat moet Mimi doen als de gebruiker op een toets drukt.

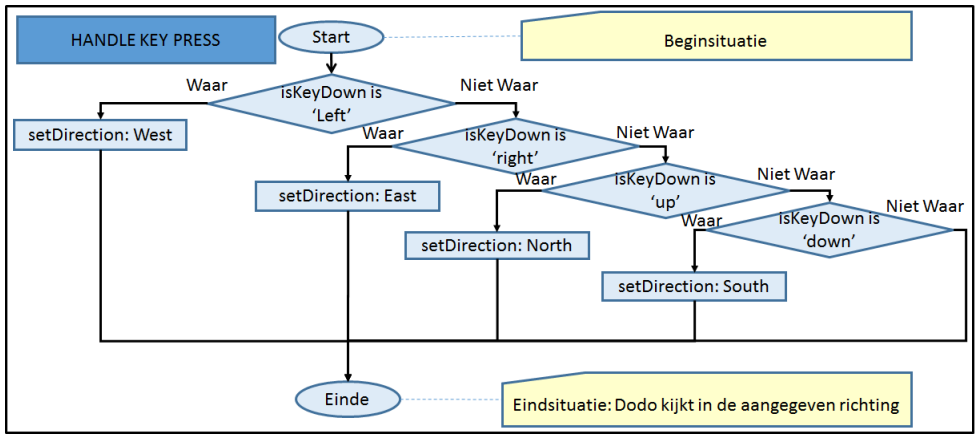
Geneste `if .. then .. else` statements
In een programma kun je meerdere gevallen testen door **`if`** .. `then` .. **`else`** genest te gebruiken. Dit is in opdracht 7 al uitgelegd met een algemeen voorbeeld. Nu geven we er een Greenfoot voorbeeld bij dat je in deze opdracht gaat gebruiken.
Voorbeeld:
Stel we willen controleren of de gebruiker één van de pijltoetsen op het toestenbord heeft ingedrukt en zo ja, Mimi vervolgens in die richting laten kijken. In Greenfoot bestaat de methode **`boolean`** `isKeyDown( String key )` waarmee je kunt testen of de gebruiker een bepaalde toets heeft ingedrukt. Als je bijvoorbeeld wil weten of de 'pijl-naar-boven' toets is ingedrukt dan gebruik je hiervoor de volgende aanroep:

```
Greenfoot.isKeyDown("up")
```

De string `"left"` hoort bij pijl-naar-links, `"right"` bij pijl-naar-rechts, en `"down"` bij pijl-naar-beneden. De volgende methode, die we `handleKeyPress` hebben genoemd, voert de gewenste aanpassing van Mimi's kijkrichting uit:

```
public void handleKeyPress() {
    if (Greenfoot.isKeyDown("left") ) {
        setDirection ( WEST );
    } else {
        if (Greenfoot.isKeyDown("right") ) {
            setDirection ( EAST );
        } else {
            if (Greenfoot.isKeyDown("up") ) {
                setDirection ( NORTH );
            } else {
                if (Greenfoot.isKeyDown("down") ) {
                    setDirection ( SOUTH );
                }
            }
        }
    }
}
```

Het bijbehorende stroomschema ziet er dan zo uit:



Figuur 1: Stroomdiagram voor `handleKeyPress`

De vele accolades en het feit dat de code zich hoe langer hoe verder naar rechts uitstrekt maakt dit soort code al snel onoverzichtelijk en daardoor ook foutgevoelig. Deze vorm van nesting kan je in Java op een ander maneier schrijven. Dit wordt ook door de meeste programmeurs gebruikt. De **`else`** en de de **`if`** worden daarbij gecombineerd. Zo komen ze samen op één regel te staan. De methode `handleKeyPress` van hierboven komt er dan zo uit te zien:

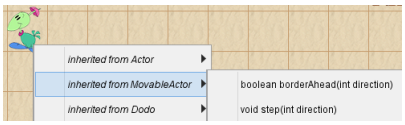
```
public void handleKeyPress() {
    if ( Greenfoot.isKeyDown( "left" ) ) {
        setDirection ( WEST );
    } else if ( Greenfoot.isKeyDown( "right" ) ) {
        setDirection ( EAST );
    } else if ( Greenfoot.isKeyDown( "up" ) ) {
        setDirection ( NORTH );
    } else if ( Greenfoot.isKeyDown( "down" ) ) {
        setDirection ( DOWN );
    }
}
```

We hebben minder regels nodig en bovendien is de structuur is veel helderder.

5 Opgaven

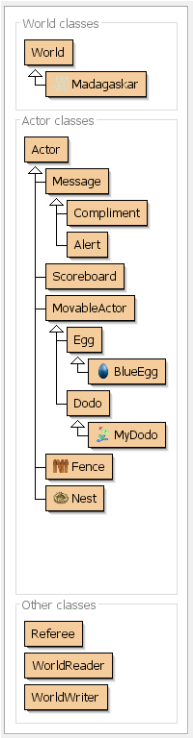
Je begint dit spel met een gloednieuw scenario. Ten opzichte van de vorige scenario's zijn er wat kleine aanpassingen. Er is een klasse bijgekomen: `MovableActor`.

In `MovableActor` zit niet echt nieuwe functionaliteit. Een aantal methodes die te maken hebben met 'verplaatsen', zoals
`void` `step(` **`int`** `direction )` en **`boolean`** `borderAhead (` **`int`** `direction )` zijn van de `Dodo` klasse naar de klasse `MovableActor` verplaatst. Dat Mimi nu deze methodes uit `MovableActor` erft is in het plaatje te zien:



Figuur 2: Methodes die Dodo erft uit `MovableActor`

Iedere `Dodo` kan verplaatst worden en hoort dus thuis in de klasse `MovableActor`. Omdat in deze opdracht eieren ook verplaatst kunnen worden, horen eieren ook thuis in de klasse van `MovableActor`. In het klassediagram hiernaast zie je dat zowel `Dodo` als `Egg` subklasses van `MovableActor` zijn. Ook een ei kan dus de methodes `step` en `borderAhead` gebruiken.



Daarnaast zijn aan de klasse `Egg` ook een paar methodes toegeveogd. De belangrijkste zijn:

`void` `push(` **`int`** `direction )`: waarmee een ei in een bepaalde richting geduwd kan worden. Deze richting wordt als parameter meegegeven.

`boolean` `canBePushed(` **`int`** `direction )`: waarmee getest kan worden of het ei in de opgegeven richting kan worden verplaatst.

5.1 Opstarten

1. Download en open het scenario `madagaskarSB-Start`
2. Compileer het scenario. Druk op *Run*.
3. Als het goed is loopt Mimi langs het hek dat de geopende wereld omheint.
4. Test met de rechtermuisknop de nieuwe ei methodes `push` en `canBePushed` beide een aantal keren uit. Plaats daarbij het ei op verschillende posities in de wereld. Werken deze methodes zoals je verwacht?
5. Vervang de code in de `act` van `MyDodo` door een aanroep van `handleKeyPress`. Deze methode vind je ook in `MyDodo` terug.
6. Compileer en run het scenario (door weer op *Run* te drukken). Wat gebeurt er? Niets? Druk dan één van de pijltoetsen op je toetsenbord in. Op welke toets(en) reageert Mimi?

5.2 Mimi reageert op pijltjestoetsen

Zoals je ziet gaat er nog van alles mis. Mimi loopt nog gewoon door het hek en over eieren heen en ze reageert nog niet goed op de pijltjestoetsen die door de speler worden ingedrukt. Dit ga je zelf in de komende opgave in orde maken. We beginnen met de bediening van Mimi.

1. Open de klasse `MyDodo` in de editor en zoek `handleKeyPress` op. Bekijk de body van deze methode. Hier wordt gebruik gemaakt van een andere hulpmethode genaamd `getNewDirection`. Bedenk waar je wat moet aanpassen zodanig dat Mimi op alle pijltoetsen reageert. Tip: bekijk het theorieblok hierboven over 'Geneste **`if`** .. `then` .. **`else`** statements' waarin de Greenfoot methode `isKeyDown` ook aan de orde komt. Maak gebruik van deze informatie.
2. Pas `MyDodo` aan en compileer het scenario.
3. Run het scenario en controleer of Mimi inderdaad op alle pijlen goed reageert. Zo niet, pas je code aan.

5.3 Mimi duwt eieren vooruit (A)

Nou gaan we ervoor zorgen dat Mimi niet meer gewoon over de eieren heen loopt maar ze vooruitduwt zodra ze er tegenaan loopt.

1. Zoals je in de uitleg zag wordt op dit moment een stap gezet zodra er een pijltoets is ingedrukt. We moeten er nu voor zorgen dat in sommige gevallen niet alléén een stap wordt gezet. Pas hiervoor `handleKeyPress` aan. Tips:
   - Bedenk eerst welke gevallen er speciaal behandeld moeten worden.
   - De klasse `Dodo` bevat al methoden die je nodig hebt om deze gevallen te onderscheiden.
   - Uiteraard zullen ook de nieuwe `Egg`-methodes van pas komen.
   - Met `eggAhead` kan Mimi nagaan óf er een ei voor haar ligt en met `getEggAhead` krijgt ze het ei te pakken. Ze kan dan eerst aan het ei vragen of het geduwd kan worden, en als dat zo, het ei vooruitduwen om er daarna zelf een stap achteraan te zetten.
2. Compileer en test jouw aanpassingen. Probeer een aantal gevallen systematisch uit. Repareer meteen de foutjes die je tegenkomt. Controleer of Mimi inderdaad aan alle eigenschappen uit onderdeel 1 voldoet.

5.4 Scorebord

We willen nog bijhouden hoeveel stappen Mimi tot nu toe gezet heeft en hoeveel eieren al in een nest liggen. We laten dit door Mimi zelf bijhouden.

1. Voeg eerst twee instantievariabelen aan `MyDodo` toe waarin deze informatie (aantal stappen gezet en aantal eieren in een nest) wordt bijgehouden. Bedenk een zinvolle naam, een type en een geschikte beginwaarde.
2. Pas de instantievariabele voor het aantal stappen aan zodra Mimi één stap zet.
3. Roep ook `updateScores` aan om ervoor te zorgen dat de veranderde situatie daadwerkelijk op het scorebord getoond wordt.
4. Dit was een makkelijk deel van de opdracht, maar om te voorkomen dat er een foutje is ingeslopen compileer en test je jouw toevoeging voordat je verder gaat.
5. En nu de tweede variabelen om bij te houden hoeveel eieren in de nest liggen. Bedenk wanneer en hoe je dit kunt nagaan. Tip: wellicht dat de `Dodo` methode **`boolean`** `nestAhead( )` van pas kan komen.
6. Roep hier ook weer `updateScores` aan.
7. Compileer en test je programma.

5.5 Ei in het nest

Als een ei nu in een nest wordt geduwd dan zie je het ei niet meer. Als het ei en nest namelijk op dezelfde plek staan dan zit het eitje verstopt achter het nest. Dat is jammer. Dit gaan we verbeteren. We passen hiervoor eerst de klasse `Nest` aan.

1. Open de klasse `Nest` in de editor.
2. Voeg aan deze klasse `Nest` een instantievariabele toe die aangeeft of het nest leeg is of niet. Zorg ervoor dat deze variabele een geschikte beginwaarde krijgt.
3. Voeg methodes om het nest te vullen en weer leeg te maken. Het effect hiervan is niet alleen dat de instantievariable een andere waarde krijgt maar ook dat het plaatje van het nest wordt aangepast. Tip: Kijk terug naar opdracht 6 opgave 'Ei uitbroeden' waar je iets vergelijkbaars hebt gedaan in de klasse `Egg`. Daarbij toonde je het plaatje van een uitgebroede ei.
4. Compileer en test jouw programma met de rechtermuisknop.
5. Pas ook jouw klasse `MyDodo` aan. Bedenk eerst wat er precies moet gebeuren en waar jouw code moet worden aangepast. Om de toestand van het nest te kunnen veranderen heb je (een verwijzing naar) het nest-object nodig. Welke methode kun je daarvoor gebruiken? Hou er ook rekening mee dat je een ei weer uit het nest kunt duwen.
6. Compileer en test je programma, dit keer ook door het scenario te runnen.

5.6 Level gehaald (A)

Het programma moet nu zelf bepalen of het spel afgelopen is. Om Mimi niet nog meer te belasten laten we deze klus aan de scheidsrechter (de referee) over.

1. Bekijk de code van de klasse `Referee`.
2. Bovenin de klasse staat een instantievariabele `myWorld` waarin een verwijzing naar de wereld wordt bijgehouden. Verderop staat de methode `updateScoreboard`. Deze methode wordt in de klasse `Dodo` aangeroepen zodra je met `updateScores` de scores aanpast. Je kunt dit, als je nieuwsgierig bent, zelf controleren door `Dodo` te openen en de methode `updateScores` op te zoeken.
3. De tweede score, genaamd `score2`, geeft aan hoeveel eieren al in een nest liggen. Als dit gelijk is aan het aantal eieren in de wereld, dan is het spel afgelopen. Voeg een methode toe die controleert of het spel is afgelopen. Roep als de test **`true`** oplevert de methode `levelFinished` aan. Tip: Om er achter te komen hoeveel eieren de wereld bevat kun je de `World`-methode `getObjects` gebruiken. Zoek in de Java documentation op wat deze methode doet en bedenk hoe je het resultaat van deze methode in jouw test kunt gebruiken.
4. Compileer en test je programma.
5. Geef Mimi een compliment als het level behaald is.

5.7 Optioneel: Oops.. undo?

Tot slot het is nog een tikkeltje onbevredigend dat, als je per ongeluk op een verkeerde pijl drukt, je in een situatie kunt komen als het spel niet meer oplosbaar is. Wat je eigenlijk zou willen is een *undo*-functie. Hiervoor moet je een aantal dingen toevoegen. We doen dit aan de klasse `myDodo`. We geven hier enkel wat globale aanwijzingen. De details moet je zelf bedenken en uitwerken.

1. Bepaal welke toets geschikt is om als *undo*-toets te gebruiken. Voeg een methode om dit af te handelen.
2. Bepaal welke informatie je moet onthouden om een stap ongedaan te kunnen maken.
3. Vergeet niet dat behalve Mimi soms ook een ei weer op de oude positie teruggezet moet worden.
4. En wat moet er gebeuren als de gebruiker meerdere stappen ongedaan wil maken?
5. Verschijnt in dat geval nog steeds de juiste waarde op het scorebord?

5.8 Optioneel: Nieuwe levels

Voeg zelf nieuwe uitdagendere levels toe aan het scenario. Laat deze testen door een medestudent om te controleren of ze oplosbaar zijn. Laat meteen bepalen wat de moeilijkheidsgraad van die level is.

6 Samenvatting

Je hebt geleerd:

- gebruikersinteractie af te handelen;
- gebruik te maken van een **`else if`**;
- wat je in de vorige opdrachten hebt geleerd zelf te gebruiken om een spel te schrijven.

7 Jouw werk opslaan

Je bent klaar met de achtste opdracht. Sla je werk op, want je hebt het nodig voor de volgende opdrachten.

1. Kies in Greenfoot 'Scenario' in het bovenste menu, en dan 'Save As ...'.
2. Vul de bestandsnaam aan met jouw eigen naam en het opgavenummer, bijvoorbeeld `Opdr8_Michel`.

Alle onderdelen van het scenario bevinden zich nu in een map die dezelfde naam heeft als de naam die je hebt gekozen bij 'Save As ...'.