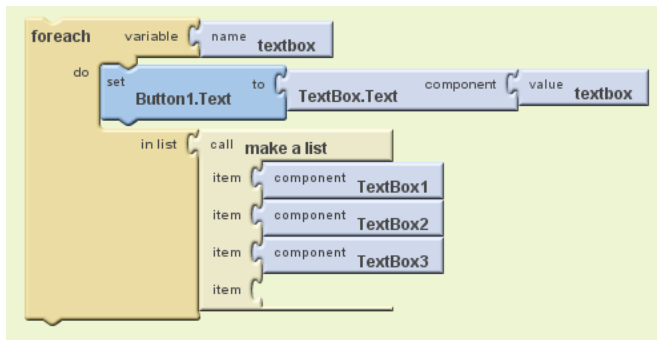


# Android Apps met App Inventor

Coen Crombach

Robin Eggenkamp

François Vonk



6 juni 2012



# Voorwoord

Dit boek is geschreven in het kader van Onderzoek van Onderwijs aan de Eindhoven School of Education, Technische Universiteit Eindhoven. Het boek is bedoeld voor gebruik als introductie tot programmeren in het vak Informatica in de bovenbouw van het voortgezet onderwijs. Aan de hand van enkele applicaties voor Android telefoons zul je de principes van programmeren leren.


## Gebruikte notaties

In dit boek worden enkele notaties gebruikt om het voor jou als lezer makkelijker leesbaar te maken. In deze sectie geven we hier een opsomming van.

**Block** Blocks zul je gebruiken om te programmeren. Blocks in *App Inventor* worden weergegeven als puzzelstukjes.

**Knop of menu** Menuopties of knoppen worden op deze manier aangegeven.

**Term** Belangrijke termen worden schuingedrukt om deze extra te benadrukken.

 **Hints** Naast de tekst worden soms belangrijke hints of tips gegeven.

**Opgaven** Opgaven worden aangeduid met een blauwe balk.



### Opgave 0.1

Hier staat de vraag.

**Uitwerking:** In sommige gevallen wordt ook een uitwerking gegeven.

**Waarschuwing** In sommige gevallen willen we je een belangrijke waarschuwing of tip geven. Deze worden voorzien van een rode balk.



### Waarschuwing

Hier staat een waarschuwing, lees deze zorgvuldig!



**Uitproberen!** Regelmatig kom je de afbeelding hier-naast tegen. Dit betekent dat je de app waar je op dat moment mee bezig bent uit mag proberen op je telefoon of in de emulator.

# Inhoudsopgave

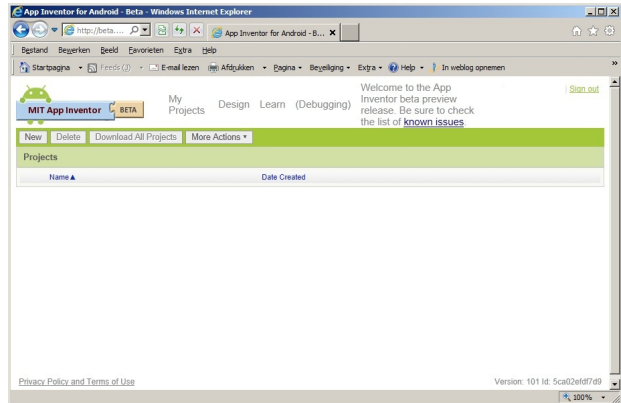
<b>Voorwoord</b>	<b>iii</b>
Gebruikte notaties . . . . .	iii
<b>1 Installatie</b>	<b>1</b>
<b>2 Ontwikkelomgeving</b>	<b>5</b>
<b>3 Mollen Meppen</b>	<b>9</b>
3.1 Willekeurig opduiken van de mol . . . .	13
3.2 Geluid toevoegen als je de mol raakt . .	14
3.3 De mol op een timer laten bewegen . .	15
3.4 Tellen van het aantal maal dat je mis slaat	17
3.5 Score toevoegen . . . . .	17
3.6 Mol sneller laten bewegen bij hoge score	18
3.7 Bonus opgaven . . . . .	18
3.8 Testen op je telefoon . . . . .	19
<b>4 Schrandere Scholier</b>	<b>21</b>
4.1 Een nieuw project aanmaken . . . . .	22
4.2 Gebruikersinterface . . . . .	22
4.3 Rekenen . . . . .	26
4.4 Verbeteren . . . . .	28
4.5 Weer rekenen . . . . .	32
4.6 Facultatieve uitbreidingen . . . . .	35

<b>5</b>	<b>Meteoor</b>	<b>37</b>
5.1	Vooraf . . . . .	37
5.2	De Orientation Sensor . . . . .	38
5.3	De raket . . . . .	40
5.4	Besturing . . . . .	42
5.5	Kantelen . . . . .	44
5.6	$x_{\max}$ . . . . .	45
5.7	Geen Orientation Sensor? . . . . .	47
5.8	Raket onderaan scherm . . . . .	49
5.9	Obstakels . . . . .	50
5.10	Meerdere meteoren . . . . .	52
5.11	Botsing . . . . .	53
5.12	En weer verder . . . . .	55
5.13	Mogelijke uitbreidingen . . . . .	56

# Hoofdstuk 1

## Installatie

De *App Inventor* ontwikkelomgeving draait in een web-browser. Om toegang tot *App Inventor* te krijgen heb je een Google account nodig. Je kunt zo'n account aanvragen op: <http://accounts.google.com/NewAccount?hl=nl>. Als je een Google account (aangemaakt) hebt kun je inloggen via de volgende link: <http://beta.appinventor.mit.edu>. Nadat je ingelogd bent zie je een scherm zoals afgebeeld in figuur 1.1.

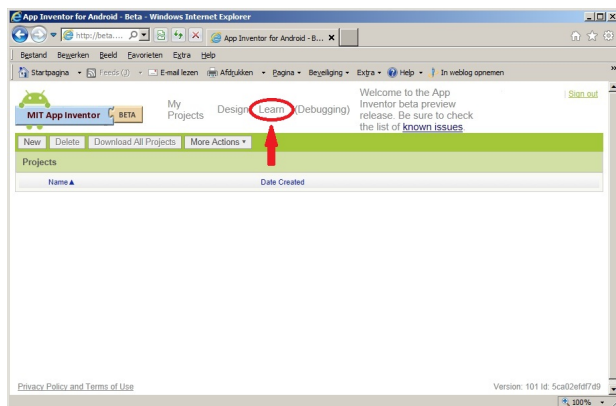


**Figuur 1.1** 'My Projects' scherm

Op school is alle software die *App Inventor* nodig heeft al voor je geïnstalleerd. Maar als je thuis aan de slag gaat is dat misschien niet zo. Daarom volgen hierna de instructies om thuis te zorgen dat de *App Inventor* ontwikkelomgeving goed werkt. Als je al een werkende omgeving hebt kun je meteen door naar het volgende hoofdstuk.

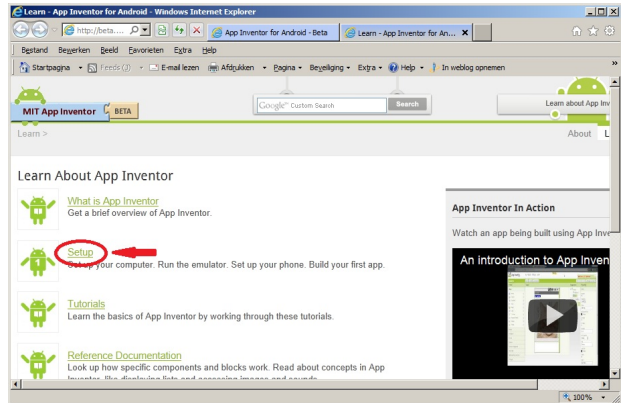
Het is nu belangrijk dat je de software installeert die *App Inventor* nodig heeft om de *Blocks Editor* en de emulator uit te voeren. Deze termen zullen je nu misschien nog niets zeggen en dat is op dit moment niet erg. Verderop in het materiaal, als je ze nodig hebt, zullen ze in detail behandeld worden. Het is nu belangrijk om op de 'Learn' link te klikken, zie figuur 1.2.





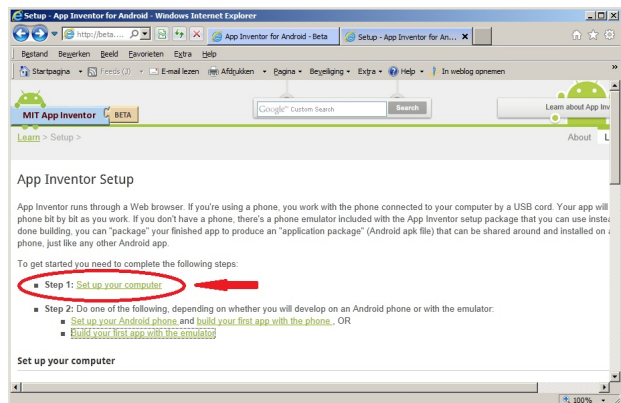
**Figuur 1.2** Locatie van de 'Learn' link

Als we dit hebben gedaan zien we het scherm zoals afgebeeld in figuur 1.3. Zoals je kunt zien zijn we op dit moment geïnteresseerd in de 'Setup' link. Via deze link komen we namelijk te weten wat we allemaal moeten installeren om te kunnen werken met alle functionaliteit die *App Inventor* ons te bieden heeft. We moedigen je echter ook aan om op de andere links te klikken en even rond te neuzen wat je daar kunt vinden. Sommige links zijn handig als je later nog eens iets op wilt zoeken.



**Figuur 1.3** Locatie van de 'Setup' link

Nadat je op de 'Setup' link hebt geklikt, zie figuur 1.3, kom je op een pagina met installatie instructies. Voorlopig hoeft je alleen 'Step 1' te doen 'Set up your computer', zie figuur 1.4. De tweede stap mag je overslaan omdat we die in dit lesmateriaal met je gaan doorlopen.



**Figuur 1.4** Locatie van de 'Set up your computer' link

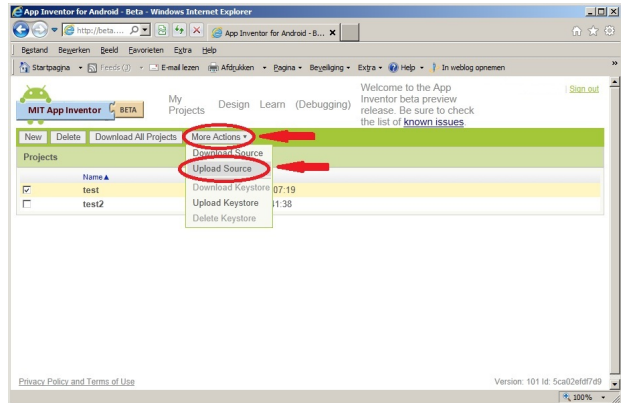
## Hoofdstuk 2

# Ontwikkelomgeving

### Waarschuwing

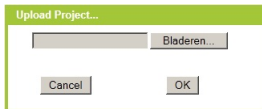
Voordat we van start gaan met *App Inventor* een kleine waarschuwing. De ontwikkelomgeving is niet altijd even snel en het gaat niet sneller door veelvuldig op links of knoppen te klikken. Sterker nog: dit zorgt er voor dat de omgeving nog langzamer wordt en dat er fouten op gaan treden. Om dit te voorkomen vragen we je om geduld te hebben en te wachten als de omgeving je daarom vraagt!

Als het goed is heb je het scherm voor je zoals afgebeeld in figuur 1.1 en heb je nog geen projecten. Het eerste dat we daarom gaan doen is een project *uploaden*. Dat wil zeggen dat we een project dat iemand anders voor ons gemaakt heeft gaan binnenhalen in de ontwikkelomgeving. Om dit te doen klikken we op de **More Actions** - knop en kiezen vervolgens voor **Upload Source**, zie figuur 2.1.



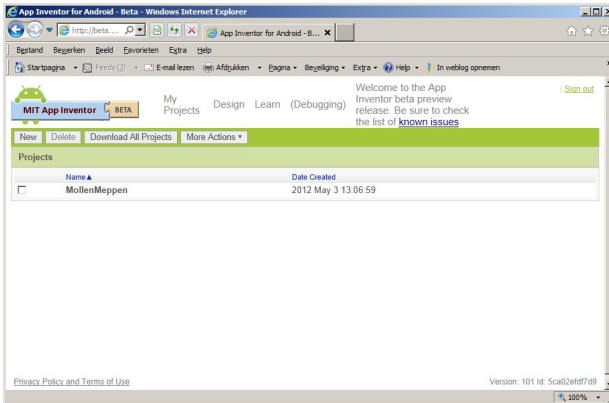
**Figuur 2.1** Locatie van 'Upload Source'

In ons geval gaan we het 'MollenMeppen' project binnenhalen. Als we op **Upload Source** hebben geklikt krijgen we de pop-up zoals afgebeeld in Figuur 2.2. Via deze pop-up kunnen we bladeren naar de locatie van het project dat we willen binnenhalen. Vraag je docent waar je het project kunt vinden. Als we het projectbestand hebben gevonden dan selecteren we het en klikken vervolgens op **OK**.



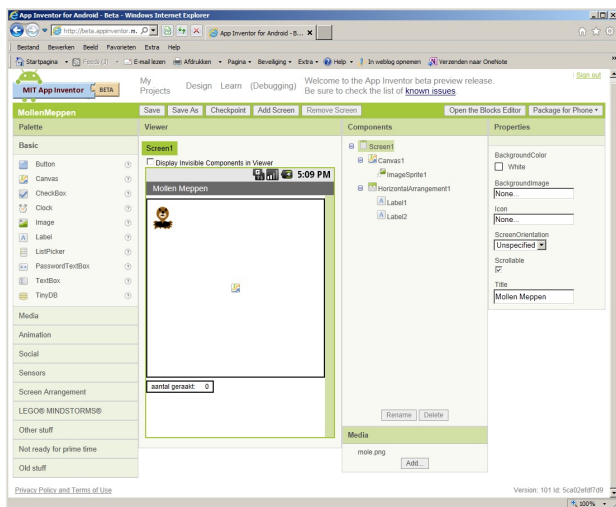
**Figuur 2.2** 'Upload Project' pop-up

Nadat we op **OK** hebben geklikt wordt het project 'MollenMeppen' in ons scherm toegevoegd, zie figuur 2.3.



**Figuur 2.3** *My Projects* scherm met MollenMeppen project

Korte tijd nadat het project is toegevoegd zal de ontwikkelomgeving automatisch naar het *Design*-scherm springen waarin we het ontwerp van de applicatie zien, zie figuur 2.4. Om weer naar het ‘My Projects’ scherm te gaan kun je deze aanklikken in de ontwikkelomgeving.



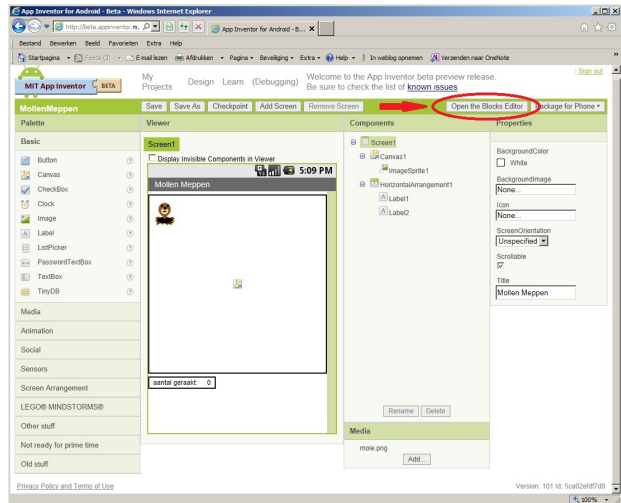
**Figuur 2.4** ‘Design’ scherm van MollenMeppen

## Hoofdstuk 3

# Mollen Meppen

We hebben zojuist ons eerste project geladen en het ontwerp ervan kort gezien. Voor we uitgebreid naar het één en ander kijken gaan we de applicatie eerst een keer uitvoeren.

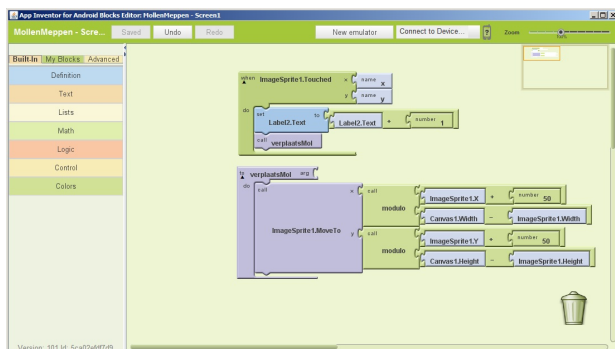
Dit doen we door vanaf de **Design** pagina de **Blocks Editor** te openen via de **Open the Blocks Editor** link, zie figuur 3.1. Nu moet je even goed opletten omdat het kan zijn dat je browser toestemming vraagt om een aantal dingen te mogen doen.



**Figuur 3.1** Locatie van de ‘Open the Blocks Editor’ link

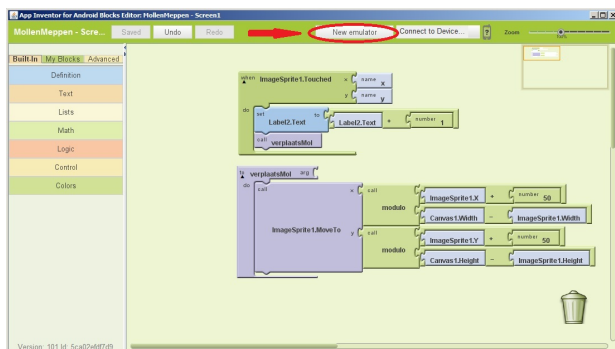
Als eerste wordt het [AppInventorForAndroidCodeblocks.jnlp](#) bestand geopend. Als je browser je hierover een vraag stelt moet je op **openen** klikken. Vervolgens wordt het bestand geladen door *Java* en kan het zijn dat je toestemming moet geven om het bestand uit te voeren. Vervolgens wordt de **Blocks Editor** geopend en zie je wat er in figuur 3.2 is afgebeeld.



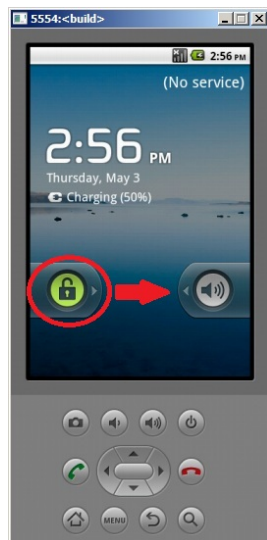


**Figuur 3.2** Blocks Editor van MollenMeppen

Om de applicatie uit te voeren moeten we eerst een emulator opstarten. Dit doen we via de **New emulator** link zoals aangegeven in figuur 3.4. Als de emulator is opgestart ziet deze er uit zoals afgebeeld in figuur 3.3. De emulator staat nu nog 'op slot', we kunnen het slot verwijderen door met de muis op het **slotje** te klikken, de muisknop ingedrukt te houden, de muis in de richting van de pijl te bewegen zoals aangegeven in figuur 3.3 en de muisknop weer los te laten.

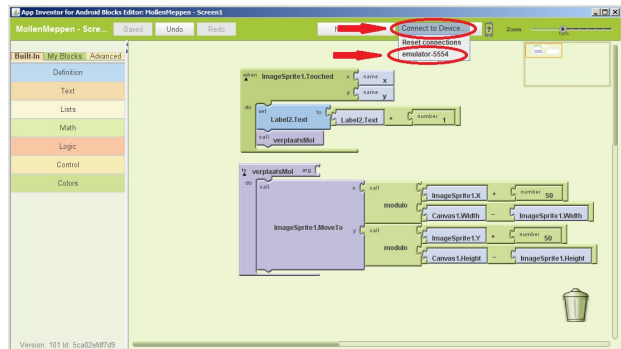


**Figuur 3.4** Locatie van de **New emulator** link

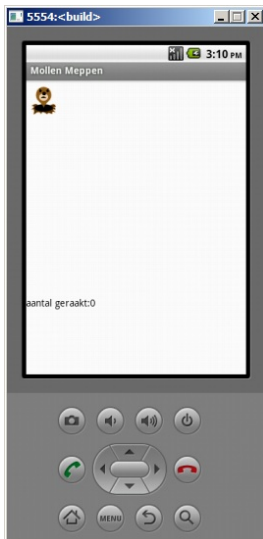


**Figuur 3.3** De emulator

Als we de emulator ‘van slot’ hebben gehaald kunnen we verbinding maken met de emulator. Dit doe we door op de **Connect to Device** link te klikken en vervolgens **emulator-5554** te kiezen, zie figuur 3.5.



**Figuur 3.5** Locatie van de **Connect to Device** link



**Figuur 3.6** De Mollen-Meppen applicatie

Nadat we op de link geklikt hebben wordt onze applicatie in de emulator geladen. Als het laden klaar is zien we wat er is afgebeeld in figuur 3.6 en kunnen we de applicatie gebruiken. Klik op de mol en kijk wat er gebeurt.

Wat er gebeurt is nog niet zo heel spannend en uitdagend maar daar kun je wat aan doen! Hieronder volgt een serie opdrachten waardoor je ‘Mollen Meppen’ tot een echt spel kunt maken. We raden je aan om na iedere opgave waarin je de code aanpast deze uit te testen op de emulator. Veel succes!

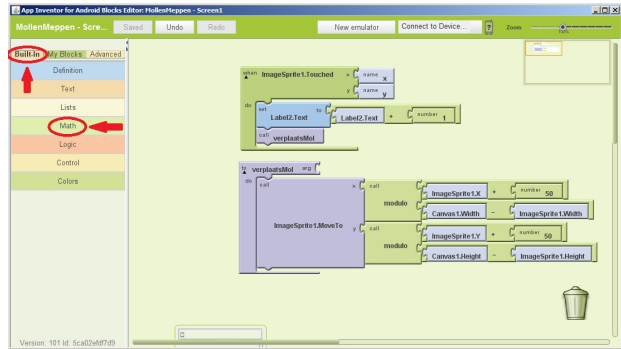
## 3.1 Willekeurig opduiken van de mol

De mol beweegt zich erg voorspelbaar op dit moment en dat maakt het spel saai. In de **Blocks Editor** kunnen we zien waarom de mol zich zo gedraagt.


### Opgave 3.1

Bekijk de code van de procedure ‘verplaatsMol’ en beredeneer waarom de mol zich verplaatst zoals je ziet wanneer je erop klikt.

Wat we hier feitelijk willen is dat de  $x$  en  $y$  positie van de mol (de *ImageSprite*) willekeurig bepaald worden. Hiervoor kunnen we een *block* vinden onder de **Built-In** tab bij **Math**, zie figuur 3.7. *Blocks* uit de **Blocks Editor** kunnen we simpelweg in het **blokkenveld** slepen en daar neerzetten of in elkaar klikken. Als je per ongeluk een verkeerd *block* hebt gebruikt kun je het verwijderen door het naar de prullenbak rechts onder in het scherm te slepen en het block los te laten.



**Figuur 3.7** Locatie van de 'Built-In' Math link

 Hint: Wat is het Engelse woord voor willekeurig?

### Opgave 3.2

Verander de code van de procedure *verplaatsMol* zodat de mol zich willekeurig verplaatst binnen het veld.

## 3.2 Geluid toevoegen als je de mol raakt

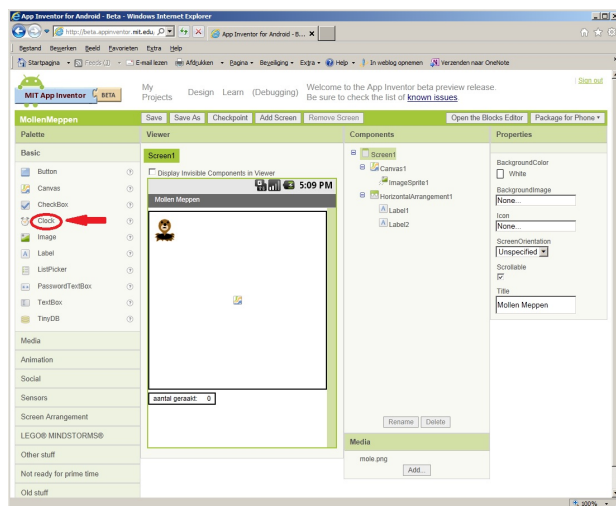
Als de mol geraakt wordt is het leuk als we dat ook horen. Gelukkig kunnen we dit programmeren in *App Inventor*. Zoek een leuk geluidje op via internet en maak de volgende opdracht.

### Opgave 3.3

Voeg een geluid toe aan het spel en speel dat geluid af op het moment dat de mol geraakt wordt.

### 3.3 De mol op een timer laten bewegen

Op het moment is het zo dat de mol zich alleen verplaatst als je hem een mep verkoopt. Het is leuker als de mol af en toe uit zichzelf ergens anders opduikt. Om dit voor elkaar te krijgen gebruiken we een zogenaamde **timer**. Deze kunnen we vinden via de **Clock** component in het **Basic** palette, zie figuur 3.8. Een component kun je in het **Viewer** gedeelte van je ontwikkelomgeving slepen. Je ziet dat de component ook toegevoegd wordt in het **Components** gedeelte van je ontwikkelomgeving. Dit is de plaats waar je componenten een andere naam kunt geven en verwijderen.

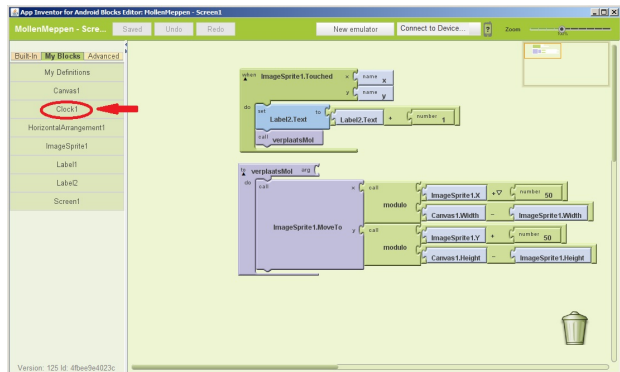


**Figuur 3.8** Locatie van de **Clock** control in het **Basic** palette

### Opgave 3.4

Voeg een **Clock** control aan het spel toe en kijk goed waar deze neergezet wordt. Beredeneer wat er gebeurt en waarom.

Nadat we de **Clock** control hebben toegevoegd kunnen we de timer programmeren. Hiervoor moeten we in de **Blocks Editor** zijn. Je weet intussen hoe je die moet openen of je hebt hem nog open staan. Onder de **My Blocks** tab staat nu een **Clock 1** link omdat we de **Clock** control hebben toegevoegd, zie figuur 3.9. Klik op de link en je ziet bovenaan een **Clock1.Timer** block.



**Figuur 3.9** Locatie van de **My Blocks** | **Clock1** link



Hint: Kijk eens bij de **My Blocks** | **My Definitions** link.

### Opgave 3.5

Voeg het timer blok toe en zorg dat de mol zich om de zoveel tijd uit zichzelf verplaatst.

### 3.4 Tellen van het aantal maal dat je mis slaat

Naast het bijhouden hoe vaak een speler de mol raakt willen we ook graag bijhouden hoe vaak de speler mislaat. In je `Components` staat een `HorizontalArrangement1` die ervoor zorgt dat je componenten horizontaal in je scherm kunt rangschikken. De componenten erin ( `Label1` en `Label2` ) komen overeen met ‘aantal geraakt’ en ‘0’ in je `Viewer`.

#### Opgave 3.6

Breid de bestaande `HorizontalArrangement1` uit met de tekst ‘aantal gemist’ en de bijbehorende teller. Breid bovendien je programma in de `Blocks Editor` uit zodat het aantal keer dat de speler mislaat wordt bijgehouden.

### 3.5 Score toevoegen

Het bijhouden van het aantal keer raak en misslaan is leuk maar een score is nog leuker. Voor iedere keer dat een speler de mol raakt krijgt hij/zij 3 punten. Voor iedere keer dat de speler de mol mist verliest hij/zij een punt.

### Opgave 3.7

Voeg een `HorizontalArrangement` toe in de `Viewer` waarin je de score informatie laat zien. Breid bovendien je programma in de `Blocks Editor` uit zodat de score wordt bijgehouden.

## 3.6 Mol sneller laten bewegen bij hoge score

Naarmate de tijd vordert en hopelijk de score hoger wordt blijft de uitdaging van het spel op dit moment gelijk. Het spel vereist helaas niet steeds meer van het reactievermogen van de speler maar dat kunnen we veranderen.

### Opgave 3.8

Zorg ervoor dat voor elke 50 punten die de speler heeft de mol 50 milliseconden sneller beweegt. Houd rekening met een grens voor de snelheid.

 Hint: Als je de `Clock` component selecteert in bijvoorbeeld de `viewer` zie je bij `Properties` een veld dat `TimerInterval` heet. Kun je iets vergelijkbaars vinden in de `Blocks Editor`?


## 3.7 Bonus opgaven

Je kunt op dit moment nog niet afgaan bij het spel. Je kunt dit veranderen door de speler af te laten gaan wanneer hij/zij onder de 0 punten zakt. Een alternatief is door levens in te bouwen in het spel en deze onder bepaalde voorwaarden af te pakken tot de speler geen levens meer over heeft.



### Opgave 3.9

Voeg naast de mol ook een andere dieren toe aan het spel. Laat deze dieren af en toe verschijnen in plaats van de mol. Deze dieren mag je niet meppen en als een speler dit toch doet verliest hij/zij een leven. Wanneer de speler geen levens meer heeft is het spel afgelopen.

 **Tip:** Laat vaker andere dieren verschijnen naarmate de speler een hogere score bereikt.

## 3.8 Testen op je telefoon

Tot nu toe heb je de app enkel getest in de emulator. Leuker is natuurlijk om de app ook op je eigen telefoon te draaien, je kunt hem dan ook thuis laten zien!

Vanuit de design omgeving kun je rechtsboven kiezen voor **Package for Phone** en vervolgens **Show barcode**. Na verloop van tijd (afhankelijk van de drukte kan dit enkele minuten duren) krijg je een venster met daarin een zogenaamde QR-code. Deze code kun je scannen met de camera van je telefoon. Hiervoor heb je een app nodig, een voorbeeld is *Qr Barcode Scanner*, je kunt deze downloaden in de *Play Store*.



Na het openen van Qr Barcode Scanner kies je voor **Scan Barcode**, je kijkt nu door je camera. Richt de camera op de barcode op het scherm. De app leest de barcode en geeft je de optie de URL die hierin verstopt is te openen in de browser. Na het openen van de browser wordt de download van een .apk bestand gestart.

Na het downloaden open je het bestand. Wat er precies gebeurt is afhankelijk van je telefoon en de ver-

sie van Android. Waarschijnlijk krijg je eerst de melding dat de installatie is geblokkeerd. Android telefoons zijn standaard ingesteld dat ze enkel applicaties vanuit de Market of Play Store kunnen installeren. Door **Onbekende bronnen** aan te vinken in het **Beveiliging** onderdeel van **Instellingen** kun je dit toestaan. Nadat je dit hebt gedaan open je de .apk opnieuw. Je krijgt nu de vraag of je de applicatie wilt installeren, je kiest voor de knop **Installeren**. Na enkele ogenblikken is de applicatie geïnstalleerd en kun je deze **Openen**.

De volgende keer dat je de applicatie via een barcode wilt installeren zal de telefoon vragen of je de applicatie wilt vervangen, dit bevestig je door op **OK** te klikken en de applicatie vervolgens op dezelfde manier te **Installeren**.

## Hoofdstuk 4

# Schrandere Scholier

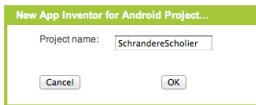
In het vorige hoofdstuk hebben we een leuk spel gemaakt, we kunnen echter ook serieuze applicaties schrijven. Waarschijnlijk vraag je jezelf wel eens af welk cijfer je moet halen voor een toets om een 8 gemiddeld te blijven staan voor een vak. In dit hoofdstuk gaan we hier een hulpmiddel voor ontwikkelen.

We zullen de applicatie stap-voor-stap opbouwen. We maken een begin met de gebruikersinterface in het **design** scherm. Daarna zullen we in de **Blocks** -editor ervoor zorgen dat de gebruiker cijfers kan invoeren waarmee we vervolgens kunnen gaan rekenen. Tot slot zorgen we ervoor dat de gegevens opgeslagen worden zodat we niet bij elke start opnieuw hoeven te beginnen.

## 4.1 Een nieuw project aanmaken

We beginnen met het aanmaken van een nieuw project. In hoofdstuk 2 heb je een project aangemaakt door een bestaand project te uploaden. Dit keer beginnen we vanaf het begin.

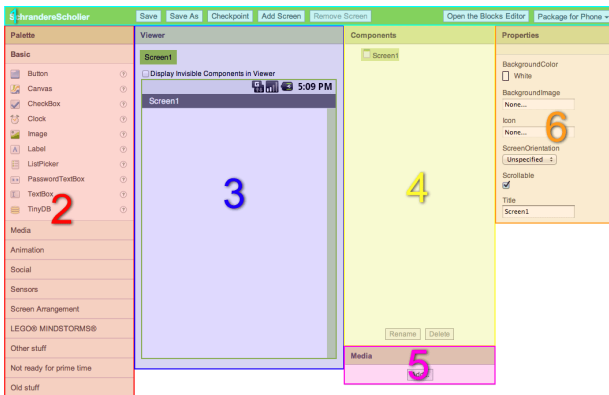
Als je nog niet in het **My Projects** scherm bent, ga hier dan nu naar toe. Klik op de knop *new* om het venster zoals is weergegeven in figuur 4.1 te zien. Vul hier de naam van het project in, in dit geval "SchrandereScholier". Na een klik op **OK** wordt je nieuwe project aangemaakt en kom je terecht in het **design** -scherm.



**Figuur 4.1** "New App Inventor for Android Project" pop-up

## 4.2 Gebruikersinterface

We beginnen met het ontwerpen van de gebruikersinterface. Voordat we hiermee aan de slag gaan kijken we naar de verschillende onderdelen van het scherm zoals je kunt zien in figuur 4.2.



**Figuur 4.2** De verschillende onderdelen van het design-scherm

- 1. De knoppenbalk** In deze balk vind je knoppen om het project op te slaan, nieuwe schermen toe te voegen en om je project op een telefoon of in de emulator uit te voeren.
- 2. Palette** Het palette is de plek waar je alle ingebouwde componenten terugvindt om te gebruiken in jouw app. Om een component te gebruiken sleep je deze naar het scherm in de Viewer.
- 3. Viewer** In de viewer kun je de gebruikersinterface ontwerpen.
- 4. Components** Dit is een overzicht van alle componenten die op dit moment in gebruik zijn. Door op een component te klikken kun je zijn eigenschappen bekijken bij Properties.
- 5. Media** In een applicatie kun je gebruik maken van bijvoorbeeld afbeeldingen, deze kun je hier toevoegen en beheren.

**6. Properties** In dit deel van het scherm pas je eigenschappen van een component aan, bijvoorbeeld de kleur.

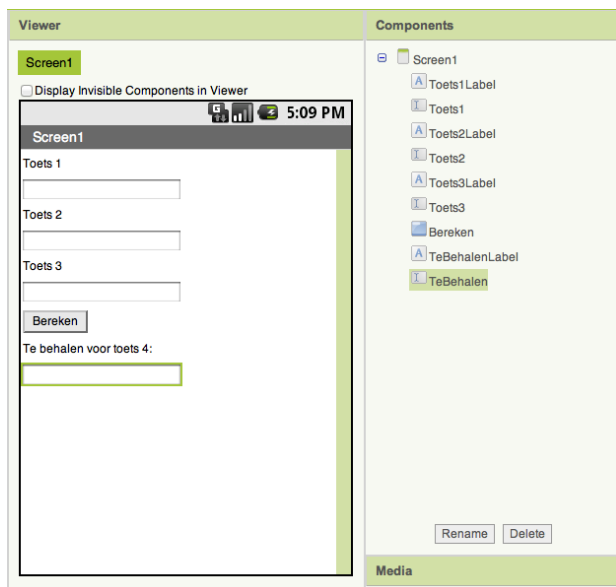
Nu je een idee hebt van de interface kunnen we deze gaan gebruiken. We beginnen eenvoudig met het toevoegen van enkele `labels`, `TextBoxen` en een `knop`. Sleep hiervoor eerst een `Label` vanuit het `Palette` naar de `Viewer`. Componenten worden automatisch uitgelijnd, het `label` komt daardoor linksboven te staan. Pas nu de tekst van het zojuist aangemaakte `Label` aan in het `Properties` paneel naar 'Toets 1'. Om onze applicatie overzichtelijk te houden pas je ook de naam van het component aan. Hiervoor selecteer je het label in het `Components` paneel en gebruik je de knop `Rename` om het label te hernoemen naar 'Toets1Label'. Om te zorgen dat er ook cijfers ingevoerd kunnen worden voeg je een `TextBox` toe. Sleep dus een `TextBox` naar de `Viewer`. Geef de `TextBox` de naam 'Toets1'. Herhaal deze handelingen totdat je drie toetsen kunt invoeren.

 Geef componenten altijd een duidelijke naam, dit maakt het programmeren makkelijker!

Zodra de gebruiker de cijfers heeft ingevoerd zul je het cijfer moeten uitrekenen wat de gebruiker moet halen voor de volgende toets om een 8 gemiddeld te halen. De berekening doen we later, maar we voegen nu al wel een knop toe en twee labels om het resultaat weer te geven. Een knop voeg je toe door een `Button` vanuit het `Palette` naar de `Viewer` te slepen. Net als bij een label kun je een text opgeven in het `Properties` paneel, vul hier 'Bereken' in. Geef de `Button` vervolgens de naam 'Bereken'.

### Opgave 4.1

Maak nu de gebruikersinterface verder af, zodat deze er hetzelfde uit ziet als in figuur 4.3.



**Figuur 4.3** Gebruikersinterface van de Schrandere Scholier

Je hebt nu de gebruikersinterface gemaakt, je kunt de applicatie nu dus al uitproberen op je telefoon of in de emulator! Je zult zien dat je al cijfers kunt invoeren, er gebeurt echter nog niks als je op de knop drukt. Daar gaan we nu iets aan doen!



### 4.3 Rekenen

☞ Een uitgebreidere uitleg vind je in hoofdstuk 2

Built-In	My Blocks	Advanced
	My Definitions	
	Bereken	
	Screen1	
	TeBehalen	
	TeBehalenLabel	
	Toets1	
	Toets1Label	
	Toets2	
	Toets2Label	
	Toets3	
	Toets3Label	

**Figuur 4.4** My Blocks van de Schrandere Scholier

☞ Blocks in App Inventor passen enkel op elkaar als de aansluitingen compatibel zijn. Dit is net als bij puzzelstukjes.

We gaan nu zorgen dat de applicatie daadwerkelijk kan rekenen. Hiervoor moeten we programmeren en dit doen we in de Blocks Editor. Deze open je via de knop Open the Blocks Editor rechtsboven aan. Het scherm is nog helemaal leeg. Aan de linker kant staan de verschillende ingebouwde codeblokken, opgedeeld in verschillende categorieën, die je kunt gebruiken. Naast de ingebouwde codeblocks zijn er ook blocks speciaal voor jouw applicatie, deze vind je onder My Blocks. Voor elk component dat je hebt aangemaakt in de gebruikersinterface kun je hier codeblocks vinden.

We willen dat er gerekend gaat worden zodra er op de knop Bereken geklikt wordt. Hier is een speciaal block voor genaamd `Bereken.Click`. Je gebruikt dit block door eerst op Bereken te klikken en vervolgens `Bereken.Click` naar rechts te slepen. Alle codeblocks die je in dit block hangt worden uitgevoerd als er op de knop geklikt wordt.

We beginnen met het verzamelen van de gegevens waarmee we straks kunnen rekenen. De cijfers moeten we uitlezen uit de TextBoxen, ook hiervoor staan er blokken klaar onder My Blocks. Het block `Toets1.Text` geeft je de inhoud van Toets1 terug. Sleep deze naar rechts om het te gebruiken. Zoals je waarschijnlijk ziet kun je nog niks met dit blok, de aansluiting komt namelijk niet overeen met de beschikbare aansluiting van `Bereken.Click`. We hebben dus nog andere blocks nodig.

We willen straks gaan rekenen, een logische plek om

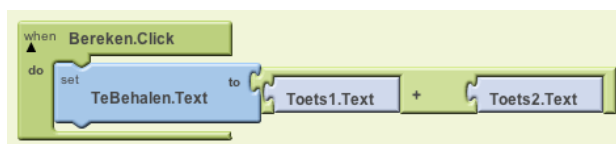


naar een passend block te zoeken is dus de categorie **Math**. Sleep bijvoorbeeld het optel block naar rechts. Je zult zien dat je hier **Toets1.Text** in kunt hangen. Niet al deze blocks hebben dezelfde aansluiting. Hoe komt dit? Deze aansluiting betekent dat het block een bepaalde waarde (bijvoorbeeld een tekst of getal, maar ook de uitkomst van een som) vertegenwoordigt, dit noemen we een *expressie*. In **Bereken.Click** passen enkel blocks die een actie vertegenwoordigen, dit wordt ook wel een *statement* genoemd.

### Opgave 4.2

Na het rekenen zullen we de uitkomst in TeBehalen moeten plaatsen. Zoek nu een block op waarmee je de uitkomst van een expressie in TeBehalen kunt zetten. Voeg dit block toe aan **Bereken.Click**. De expressie die weergegeven moet worden is de som van Toets1 en Toets2.

**Uitwerking:** Het programma zou er nu uit moeten zien zoals in figuur 4.5.

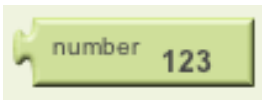


**Figuur 4.5** Uitwerking

Voordat we verdergaan: probeer uit of het programma werkt zoals je verwacht. Als het werkt dan wordt het tijd om ons programma uit te breiden. Er vanuit gaande dat alle cijfers dezelfde weging hebben, hoe kunnen we dan



het cijfer berekenen dat de gebruiker nodig heeft om een 8 te staan? De som van alle vier cijfers moet samen minimaal  $8 * 4 = 32$  zijn. Het benodigde cijfer is dus gelijk aan 32 min het totaal van de al behaalde punten.

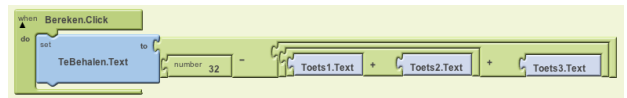


**Figuur 4.6** Constant getal

### Opgave 4.3

Implementeer de bovenstaande berekening en zet het antwoord in TeBehalen. Je hebt hiervoor, naast de al bekende blocks, ook het min-block nodig en een constant getal. Een constant getal geef je aan met het block in figuur 4.6.

**Uitwerking:** Het programma zou er nu uit moeten zien zoals in figuur 4.7.



**Figuur 4.7** Uitwerking



Als je dit programma uitprobeert met een aantal combinaties van cijfers zullen je merken dat het nog niet optimaal werkt. Maar het werkt, en dat met maar enkele blocks! In de volgende paragraaf zullen we deze applicaties verder gaan uitwerken. Je zult daarbij meer zelfstandig aan de slag gaan met *App Inventor*.

## 4.4 Verbeteren

De eerste verbetering die we zullen doorvoeren is het toevoegen van meer cijfers, waarschijnlijk heb je geen

enkel vak waarbij je maar vier toetsen krijgt.

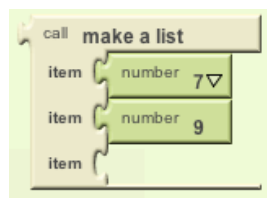
### Opgave 4.4

Zorg ervoor dat er vier cijfers kunnen worden ingevoerd, waarna het vijfde cijfer wordt berekend. Noteer op welke plaatsen je een aanpassing hebt moeten maken.

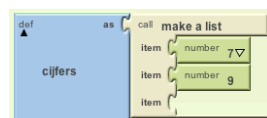
Als het goed is weet je nu precies welke wijzigingen je moet maken om een nieuwe toets te voegen. Zouden we dit kunnen automatiseren? Dan kunnen we de gebruiker zelf toetsen laten toevoegen. Uiteraard is dit mogelijk! De makkelijkste oplossing zou zijn om de gebruiker nieuwe velden te laten toevoegen, dit is echter niet mogelijk in *App Inventor*. Om dit probleem toch te kunnen oplossen heeft *App Inventor* de mogelijkheid van lijsten. We kunnen het probleem nu implementeren door gebruik te maken van één *TextBox* om cijfers toe te voegen aan de lijst. Deze lijst kan vervolgens worden weergegeven en gebruikt om te rekenen. Dit gaan we in deze sectie implementeren.

Allereerst maken we in de *Blocks Editor* de lijst aan. Je doet dit door het **make a list** block (uit het *lists* palette) het werkblad op te slepen. Zoals je ziet kun je in dit block items toevoegen, dit kan bijvoorbeeld een **number** zijn. Een voorbeeld zie je in figuur 4.8.

Om de lijst te kunnen gebruiken in de rest van de code plaatsen we de lijst in een *variabele*. Hiermee geven we de lijst een naam zodat we hier vanuit andere blocks naar kunnen verwijzen. Je maakt een variabele aan via het **def** block in de categorie **definitions** zoals je kunt zien in figuur 4.9.



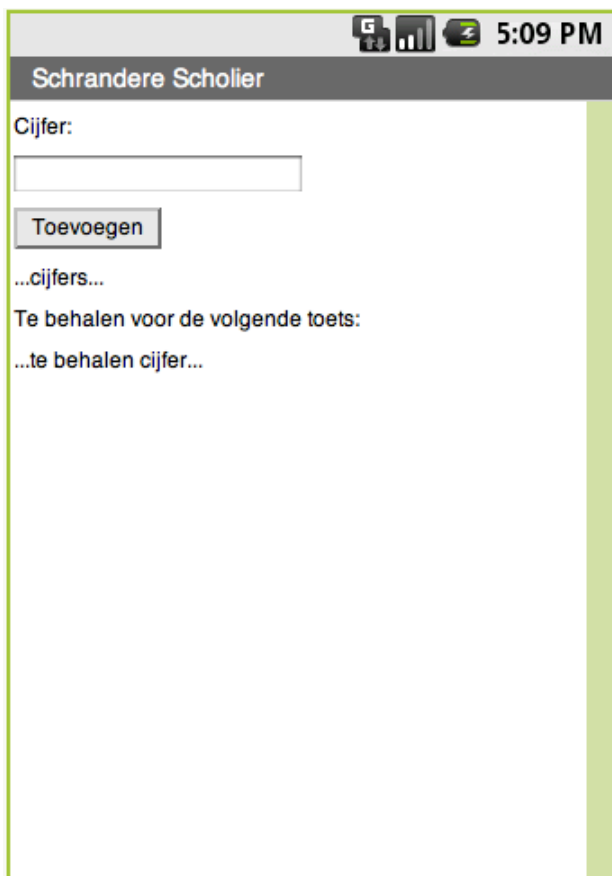
**Figuur 4.8** make a list block



**Figuur 4.9** De lijst in een variabele

### Opgave 4.5

De interface kunnen we nu aanpassen. Maak de interface zoals is weergegeven in figuur 4.10. Er staan twee labels in het ontwerp waarvan we later de tekst gaan uitpassen vanuit de Blocks editor. Denk ook nu aan een duidelijke naamgeving van de componenten.




**Figuur 4.10** Nieuwe gebruikersinterface

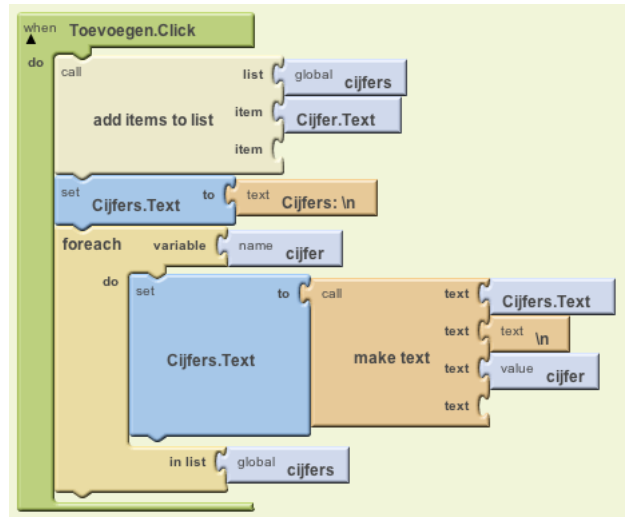
Ga nu weer naar de Blocks editor. Als je de knop enkel hebt hernoemd heb je hier nog een block `Toevoegen.Click` staan. Indien dit niet het geval is voeg je deze toe vanuit `My Blocks`. In deze procedure moeten we het nieuw ingevoerde cijfer toevoegen aan de lijst. Een item toevoegen aan de lijst kun je doen via `add items to list` in het `Lists` palette. Dit block heeft twee blocks nodig als parameter, namelijk een list en een nieuw item. Bij list voeg je het `def` block toe welke je kunt vinden bij `My definitions` binnen het `My Blocks` palette, bij item voeg je een block `Cijfer.Text` toe.

Om nu de ingevoerde cijfers weer te geven op de daarvoor bedoelde plaats moeten we nieuw soort block gebruiken, de `foreach`. Deze *loop* gaat alle items uit een list langs en voert voor elk van deze items een actie uit. In dit geval kunnen we dit gebruiken om de cijfers in de list allemaal onder elkaar te zetten. Hoe je dit doet zie je in figuur 4.11.

### Opgave 4.6

Implementeer het `Toevoegen.Click` block zoals is aangegeven in figuur 4.11 en probeer te doorgronden hoe dit werkt.

 Weet je wat de `\n` betekent in deze implementatie?



**Figuur 4.11** Implementatie Toevoegen.Click



Probeer de app nu uit op je telefoon of in de simulator en controleer of je de code goed hebt begrepen.

## 4.5 Weer rekenen

Onze oude methode om het benodigde cijfer te berekenen werkt nu niet meer, daar moeten we iets aan doen. Omdat we alle cijfers in een lijst hebben staan moeten hier weer een loop block voor gebruiken. Om te voorkomen dat onze code onoverzichtelijk wordt zullen we de berekening in een aparte *procedure* laten plaatsvinden. Een procedure is een groepje statements die worden uitgevoerd als je de procedure aanroept. Op deze manier kun je statements die bij elkaar horen groeperen. Een procedure kan ook een resultaat teruggeven.

Een nieuwe procedure kunnen we aanmaken door het `procedureWithResult` block, te vinden in het

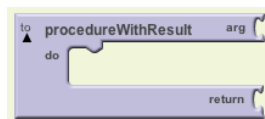
**Definition** palette, naar het canvas te slepen. Je kunt in op drie plekken blocks aanhangen. Allereerst bij 'arg', hier kun je een argument meegeven aan de procedure, dit zullen we hier nog niet gebruiken. Bij 'do' voeg je de acties toe die moeten gebeuren. Bij return geef je een waarde die wordt teruggegeven nadat deze procedure is uitgevoerd, in ons geval zal dit het te behalen cijfer zijn.

In het 'do' gedeelte van de procedure kunnen we hetzelfde `foreach` block gebruiken als in `Toevoegen.Click`. De naam van de variabele zullen we wel aan moeten passen, aangezien elke naam maar eenmaal gebruikt mag worden.

We beginnen met het optellen van alle resultaten. Dit kunnen we doen op een vergelijkbare manier als het aan elkaar plakken van de teksten, zoals we gedaan hebben in figuur 4.11. We gebruiken nu echter hetzelfde `+` block als we eerder hebben gebruikt. Het tussenresultaat sla je op in een nieuw aan te maken variabele zoals is te zien in figuur 4.13. Ook voor het eindresultaat maak je een variabele aan genaamd `teBehalen`. Gebruikmakend van de som kun je nu als volgt het te behalen cijfer berekenen:

$(aantalOpgegevenCijfers + 1) * 8 - som$

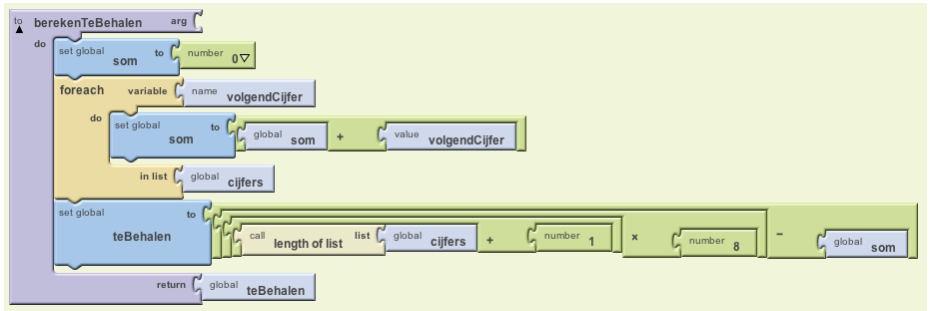
Het resultaat van deze berekening moet je teruggeven door het block `teBehalen` aan 'result' te hangen.



**Figuur 4.12** procedureWithResult block



**Figuur 4.13** som variabele



**Figuur 4.14** Implementatie van berekenTeBehalen

### Opgave 4.7

Implementeer de procedure voor het berekenen van het te behalen cijfer volgens de bovenstaande beschrijving.

**Uitwerking:** Als het goed is ziet de procedure er nu uit zoals in figuur 4.14 is getoond.



**Figuur 4.15** Aanroepen van de bereken procedure



De laatste stap is nu een klein stukje toevoegen zodat het te behalen cijfer wordt weergegeven. Dit doe je met de blocks in figuur 4.15. Voeg deze blocks toe aan **Toevoegen.Click** en test de app uit!

### Tip

Wellicht valt je op dat er ook cijfers in de lijst staan die je niet via de app hebt ingevoerd? Als dit zo is dat heb je in de Blocks Editor nog invulling staan bij het aanmaken van de lijst. Haal deze weg en probeer het opnieuw!



## 4.6 Facultatieve uitbreidingen

Je hebt ondertussen waarschijnlijk al een hoop uitbreidingen op deze app bedacht. Voeg een aantal van de onderstaande functies toe, of bedenk een eigen functie in overleg met jouw docent.

- Een 8 gemiddeld, Waarom geen 9? Laat de gebruiker zelf kiezen welk cijfer hij gemiddeld wil staan.
- Niet alle toetsen tellen even zwaar, zorg ervoor dat de gebruiker het gewicht van een toets kan bepalen.
- Geef de app een mooier uiterlijk.
- Zorg ervoor dat de cijfers worden bewaard bij het opnieuw opstarten van de app
- Maak de app geschikt voor meerdere vakken.
- Maak het onmogelijk om foutieve cijfers in te voeren.
- Jouw idee...



# Hoofdstuk 5

## Meteoor

### 5.1 Vooraf

In dit hoofdstuk gaan we een simpel spel zelf maken. We gebruiken hiervoor de `Orientation Sensor` van de mobiel, die aangeeft of het apparaat rechtop gehouden wordt of gekanteld is. We bouwen het stapje voor stapje op.

Allereerst verkennen we een stukje van de `Orientation Sensor`. Dan gaan we hiermee een zogenaamde `ImageSprite` aansturen, dat is een plaatje dat je makkelijk over het scherm kunt bewegen. In dit spel wordt de `ImageSprite` gebruikt voor een raket. De raket wordt bestuurd door de telefoon naar links of rechts te draaien.

Ook worden aanwijzingen gegeven hoe je het zonder de `Orientation Sensor` kunt maken, mocht je mobiel niet zo'n sensor hebben of voor het geval je aan-

gewezen bent op de *emulator* van *App Inventor*. Volgens zien we hoe we meteorieten over het scherm kunnen laten bewegen die ontweken moeten worden. Door te detecteren als de raket tegen een meteoriet botst weet het programma wanneer je af bent.

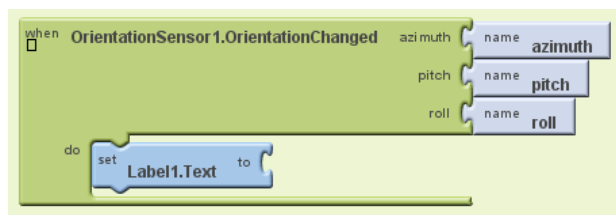
## 5.2 De Orientation Sensor

Allereerst hebben we in *App Inventor* een nieuw project nodig. Wij noemen het project 'meteor' maar je kunt zelf als je wilt een andere naam kiezen. Het eerste wat we doen is het zichtbaar maken van wat de sensor 'voelt'. Daarvoor gebruiken we `labels`. Sleep drie `labels` (Uit het `basic palette`) op het screen. Het is een goed gebruik onderdelen van je programma meteen een duidelijke naam te geven, maar deze labels gebruiken we alleen om de waarden van de sensor zichtbaar te maken, ze verdwijnen verderop weer, dus we hebben meteen een uitzondering op de regel.

In het palette `Sensors` vind je de `OrientationSensor`. Sleep ook deze op het screen. Er komt `OrientationSensor1` te staan onder het scherm onder de tekst: `Non-visible components`. Tijdens het uitvoeren van het programma is deze sensor namelijk niet zichtbaar, maar tijdens het ontwikkelen willen we natuurlijk wel ergens aan kunnen zien dat de sensor er staat.

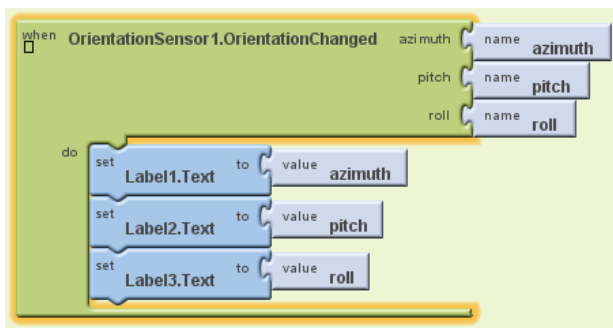
Open de `Blocks Editor` met de knop `Open the blocks editor`. Klik links op de tab `My Blocks`, dan in het rijtje eronder op

`Orientation Sensor1`, waardoor de mogelijkheden die we met dit component hebben zichtbaar worden. Eén ervan is een block met daarin de tekst: `OrientationSensor1.OrientationChanged`. Dit is een event (gebeurtenis) dat optreedt als je de mobiel draait. Als het event optreedt worden de codeblocks die in dit block geplaatst worden uitgevoerd. Sleep het block naar rechts op het programmaveld. Bovenaan het block aan de rechterkant staan drie *parameters* met de namen *azimuth*, *pitch* en *roll*. Klik onder `MyBlocks` op de naam van het eerste label, en sleep dan het block met de tekst `set Label1.Text to` in de opening in `OrientationSensor1.OrientationChanged`.



**Figuur 5.1** Het begin is er...

We willen in dit `label` de waarde laten zien van de eerste parameter: *azimuth*. Deze is bereikbaar via tab: `My Blocks` | `My Definitions`: sleep `value azimuth` naar het programmeerveld, klik hem met het *puzzel-blobje* aan de `set Label1.Text to` vast. Doe hetzelfde voor het tweede en derde label met de waarden voor *pitch* en *roll*.



**Figuur 5.2** waarden



Zet het programma op de mobiel en start het op. Draai het apparaat in verschillende richtingen en kijk naar de getallen. De waarde van *roll* is 0 als je het toestel waterpas houdt. Als je het naar links overhelst geeft dit getal aan hoeveel graden (90 als je toestel op zijn kant houdt), naar rechts ook maar dan met een min ervoor (−90 als je het toestel op zijn kant houdt). Met wat fantasie kun je het programma dat we nu hebben al gebruiken als een soort waterpas.

 **Hint:** Je kunt als je er niet uitkomt op internet de woorden *azimuth* en *pitch* nazoeken

### Opgave 5.1

Probeer zelf eens uit te vinden wat de andere twee getallen aangeven!

## 5.3 De raket

We gaan nu een raket maken die we gaan besturen met de *roll*-waarde. In *App Inventor* kunnen de **Labels** voor *pitch* en *azimuth* weg. Selecteer ze één voor één

en klik daarna op de button *Delete*. Als je met *alt+tab* naar de **block editor** gaat zie je dat de bijbehorende blocks ook verdwenen zijn.

Onder palette **Animation** vind je een **ImageSprite**. Deze willen we op het scherm laten zien. Een **ImageSprite** heeft echter een zogenaamd **Canvas** nodig om getoond te worden op een scherm. Sleep daarom een **Canvas** (palette **Basic**) op je **Screen**. Klik op het **Canvas** zodat het geselecteerd is. In de rechterkolom op het scherm (onder **Properties**) zie je een aantal eigenschappen (inderdaad: *properties* in het engels).

De bovenste is de **BackGroundColor**. Door op de kleur te klikken kun je een andere kleur kiezen. Ook met de andere waarden kun je een beetje experimenteren naar behoefte. Het meest interessant voor nu zijn echter de onderste twee: **Width** en **Height**. Stel beiden in op: *'Fill parent...'*. Je ziet dat de breedte zo breed wordt als het scherm is, echter de hoogte blijft klein. Selecteer onder **Components** het **Screen1** en zet **Scrollable** op *uit*, dan zal het **Canvas** gelijk hoger worden.

Daarmee zijn we klaar om de **ImageSprite** (palette **Animation**) op het **Canvas** te slepen. Zet hem ongeveer midden op het scherm. Dit wordt de raket. In de kolom **Components** helemaal onderin staat een balkje **Media**, met een knop **Add...**. Gebruik de *browse*-button om een plaatje van een raket toe te voegen. Wij raden aan er voor nu een te nemen uit de directory plaatjes (bijvoorbeeld 'raket2.jpg'), maar als je een beetje handig bent kun je zelf een ander (niet te

groot) plaatje ervoor in de plaats zetten. Experimenteer en leer...

Selecteer de `ImageSprite` en kies rechts bij de `Properties` voor `Picture` het zojuist toegevoegde plaatje. Op het `Canvas` kun je nu je raket bewonderen.

## 5.4 Besturing

Laten we zeggen dat we als de telefoon 45 graden naar links helt ( $roll = 45$ ) de raket links op het scherm ( $x$ -positie is dan 0) moet staan, bij 45 graden naar rechts ( $roll = -45$ ) rechts op het scherm. Rechts op het scherm wil zeggen dat de  $x$ -coördinaat bijna gelijk is aan de breedte van het Canvas (`Canvas.Width`), echter de breedte van de raket moet daar vanaf gehaald worden. Ofwel: de maximale  $x$ -coördinaat is

$$Canvas.Width - ImageSprite.Width$$

Aangezien de waarde van deze expressie constant is slaan we deze op in een variabele, die we noemen:  $xmax$ . We hebben dan dus twee waarden van  $roll$  en twee bijbehorende waarden van de  $x$ -coördinaat:

Roll	$x$ -coördinaat
-45	0
45	$xmax$

Van wiskunde ken je het *lineaire verband*: we hebben twee waarden voor  $roll$  en willen een berekening (lees: functie) die de bijbehorende  $x$ -coördinaat uitrekent. Bij wiskunde zou je het opschrijven als:



$x$	$f(x) = ax + b$
-45	0
45	$xmax$

Raak niet in de war komt doordat hier de  $x$  aan de linkerkant staat! We kunnen de technieken uit de wiskunde mooi gebruiken om de formule op te stellen. Dit kan op verschillende manieren, je hebt er bij wiskunde vast wel een geleerd. Aangezien  $-45$  en  $45$  even ver aan beide kant van de  $y$ -as liggen weten we dat voor  $x = 0$  (dus ertussenin) de  $f(x)$  ook tussen  $0$  en  $xmax$  (dus op  $0.5 \times xmax$ ) ligt, ofwel:

$$f(0) = a \times 0 + b = 0.5 \times xmax$$

waaruit volgt dat:

$$b = 0.5 \times xmax$$

dus de gezochte functie ziet er uit als:

$$f(x) = a \times x + 0.5 \times xmax$$

Door een bekende  $x$  in te vullen kunnen we de waarde van  $a$  berekenen.

### Opgave 5.2

Laat zien dat hier uitkomt:

$$a = xmax/90$$

dus we weten:

$$f(x) = xmax \times x/90 + 0.5 \times xmax$$

☞ In de wiskunde wordt  $a \times x$  afgekort tot  $ax$ , maar in de informatica is dat geen handige afspraak omdat variabelen vaak een naam krijgen langer dan één letter.

of (controleer dat dit hetzelfde betekent):

$$f(x) = (x/90 + 0.5) \times xmax$$

Terugvertaald naar de notatie met *roll* en de *x*-coördinaat staat er dan:

$$f(roll) = (roll/90 + 0.5) \times xmax$$

## 5.5 Kantelen

Terug naar de situatie: we waren aan het bepalen wat er moest gebeuren als het apparaat wordt gekanteld, ofwel als event `OrientationSensor1.OrientationChanged` optreedt. De waarde van *roll* krijgen we als parameter binnen en we willen dus de *x*-coördinaat van de raket zetten op:

$$(roll/90 + 0.5) \times xmax$$

ofwel

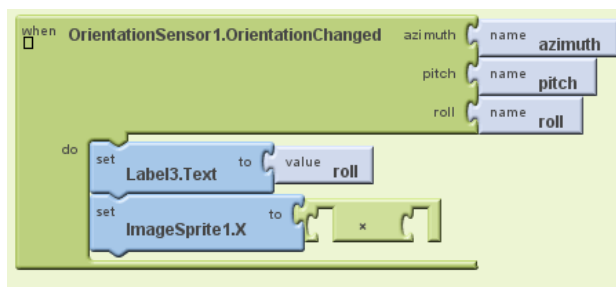
$$(roll/90 + 0.5)$$

wat een optelling is van 0.5 en

$$roll/90$$

Dit verkrijgen we door een deling te doen met de waarde van *roll* door 90. We hebben dus een *vermenigvuldiging* van een *optelling* van een *deling* (een hele mond vol) en de *uitkomst* willen we plaatsen in de *x*-coördinaat van de `ImageSprite`, dus in het block `OrientationSensor1.OrientationChanged` voegen we een block toe uit `My Blocks`, `ImageSprite1`,

namelijk *Set ImageSprite1.x to*. In het 'to'-vak voegen we allereerst een block toe voor de vermenigvuldiging. Onder tab **Built-in**, als we op **Math** klikken, vinden we een rij blocks die met rekenen te maken hebben. De eerste hiervan die we nodig hebben is het block voor de vermenigvuldiging: **×**, en er zitten twee openingen in waar de twee waarden of expressies in moeten die vermenigvuldigd moeten worden.



Figuur 5.3 positie

## 5.6 xmax

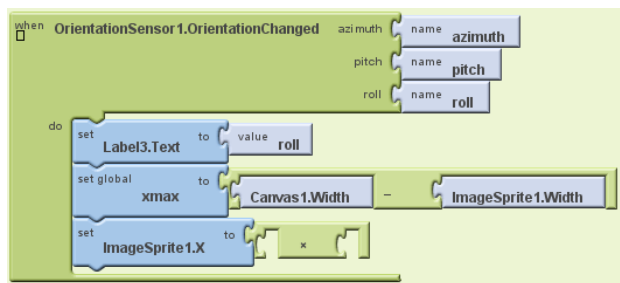
We zijn nu op het punt aangekomen dat we de variabele *xmax* nodig hebben. De definitie van een variabele gebeurt ergens op het programmeerveld: **Built-in** | **definition** | **def variable as** slepen we op het programmeerveld. Het woord 'variable' moeten we vervangen door de naam van de variabele, in ons geval dus *xmax*. Klik op *variable* en typ *xmax* gevolgd door *enter*. Hiervan wordt eenmalig de waarde gezet op *Canvas.Width - ImageSprite.Width*, dus een verschil (plaats een block met een **-** aan **def xmax as**, echter dat mag niet op de plek van de *DEFinitie*, daar



**Figuur 5.4** global  
xmax

moet eerst een constante waarde in de variabele gezet worden, bijvoorbeeld 0.

In het block waar we de waarde nodig hebben gaan we de berekening doen en het resultaat zetten we in de variabele. Sleep een block `set global xmax to` (uit *My Blocks | My Definitions*) naar het block `OrientationSensor1.OrientationChanged`, onder het commando dat de waarde van roll in de textbox zet. In `set global xmax to` komt een block `-`. In de linkse opening komt `Canvas.width` (het block vind je onder *My blocks | Canvas1*), in de rechtse `ImageSprite1.width` (vind dit onder *My blocks* in `ImageSprite1`). De waarde van `xmax` wordt nu berekend



**Figuur 5.5** Introductie global xmax

en we gaan nu de waarde van deze variabele gebruiken in de vermenigvuldiging die we in `OrientationSensor1.OrientationChanged` aan het opbouwen waren. Aangezien  $a \times b$  gelijk is aan  $b \times a$  mag je zelf kiezen of je in de linker- of in de rechter opening het `global xmax`-block zet (dat je kunt vinden onder *my blocks | my definitions*).

In de andere opening komt de optelling van 0.5 en de breuk, ook hier maakt niet uit welke aan de rechterkant staat. De 0.5 krijg je door onder **Math** een **number 123** -block te nemen en na het plaatsen op de 123 te klikken om dit te veranderen in 0.5 . In het andere gat moet de deling  $roll/90$  komen. Zoek deze blocks zelf (kun je zelf bedenken waar ze te vinden of moet je eerst alle categorieën doorklikken?).

Let op: De volgorde in de breuk maakt wel uit.

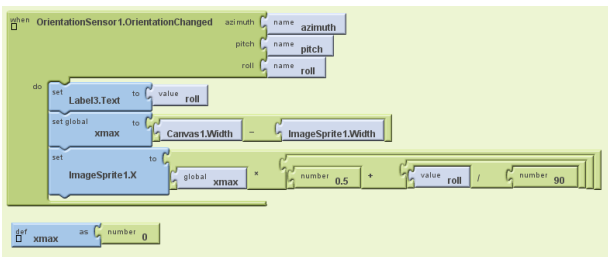
Merk op dat de blocks meteen werken als een soort *haakjes*: het gebouwde betekent dus echt

$$x_{max} \times (0.5 + (roll/90))$$

en *niet* bijvoorbeeld

$$((x_{max} \times 0.5) + roll)/90$$

Probeer het gebouwde programma dat je tot nu toe hebt eens uit: hou de mobiel schuin en kijk wat de raket doet.



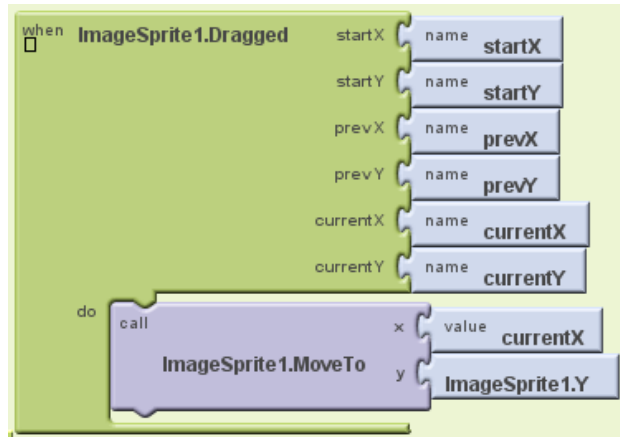
**Figuur 5.6** Je kunt nu de raket besturen

## 5.7 Geen Orientation Sensor?

Als je geen mobiel hebt loop je hier wel tegen een uitdaging aan: de **Orientation Sensor** test je niet zo-

☞ De y-waarde moet blijven wat ie was!

maar uit in een *emulator*. Om dit op te lossen hebben we het volgende alternatief bedacht: In de **block editor**, in de categorie **ImageSprite1** onder **My Blocks** vind je een block **ImageSprite1.Dragged**. Sleep dit block eens op het programmeerveld en kijk eens of je snapt wat het doet: hiermee is het mogelijk de raket te gaan besturen door met je vinger te *dragen*.



**Figuur 5.7** Drag de raket

Als je beide manieren om de raket te sturen tegelijk ter beschikking hebt geeft dat een vreemd effect: je *dragt* de raket bijvoorbeeld naar links en hij gaat daar ook naar toe, maar zo gauw je de mobiel ook maar enigszins beweegt springt de raket weer terug naar de plek die door de oriëntatie bepaald wordt. Als je dit irritant vindt kun je één van de twee uitzetten. Eén manier waarop dit kan is door op block **OrientationSensor1.OrientationChanged** óf op block **ImageSprite1.Dragged** te *rechtermuisklikken*

en dan in het *contextmenu* dat verschijnt **Deactivate** te kiezen. Vooral tijdens het ontwikkelen kan het soms handig zijn iets tijdelijk uit te kunnen zetten.

### Opgave 5.3

(optioneel) Als je heel handig bent kun je misschien zelfs maken (bijvoorbeeld met behulp van een **CheckBox** dat de gebruiker op de mobiel kan kiezen welke van de twee manieren hij wil gebruiken.

## 5.8 Raket onderaan scherm

We hebben nu een raket gemaakt die we kunnen besturen, we komen echter nog geen obstakels tegen op onze weg door het heelal. Daar gaan we nu verandering in brengen.

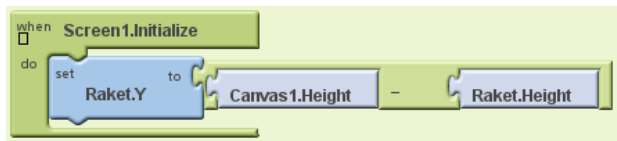
Maar eerst moet de raket onderaan op het scherm staan, dat wil zeggen dat de *y*-coördinaat maximaal moet zijn, maar wel zo dat de raket nog op de canvas staat en zichtbaar is: de waarde van de *y*-coördinaat moet daarom zijn:

$$canvas.height - imagesprite1.height$$

We hebben één belangrijke vuistregel bij het programmeren tot nu toe niet goed toegepast: onze raket heeft namelijk de naam *ImageSprite1*, niet bepaald een duidelijke naam. Zo gauw er meerdere objecten in beeld zijn kan dat natuurlijk niet meer. We hernoemen daarom **ImageSprite1** naar **Raket**. Selecteer hiervoor in

*App Inventor* 'ImageSprite1' (onder **Components**) om het te selecteren. Een stukje eronder zie een button **Rename**. Klik er op en hernoem de raket naar **Raket**. In de **Blocks Editor** kun je controleren dat alles er nog staat maar dan met de aangepaste naam.

Aangezien de raket altijd op dezelfde hoogte blijft hoeven we deze waarde maar 1 keer te zetten, namelijk bij het initieel opzetten van het scherm. We boffen, want tijdens het initieel opbouwen (in computertermen *initializing*) van het screen worden de commando's uit het block **Screen1.initialize** (zie **My blocks** | **Screen1**) uitgevoerd. Zet hier een **Set Raket.Y to** -block in. Als je dit niet weet te vinden, zoek dan enkele bladzijden terug waar de **ImageSprite1.X** vandaan kwam. Bouw hier de volgende formule op:



**Figuur 5.8** Zet Raket onder in scherm



## 5.9 Obstakels

De raket staat nu dus onderaan het scherm. Sleep uit het palette **Animation** een **Ball** op het **canvas**. Een **Ball** lijkt qua mogelijkheden veel op een **ImageSprite**, maar is zo rond als een bal. Je kunt het plaatje niet zelf uitkiezen. Hernoem (**Rename**)



Ball1 tot Meteor .

Als je heel precies het verschil tussen een ImageSprite en een Ball wilt weten kun je in het palette op de vraagtekens rechts van de ImageSprite en Ball klikken. In het algemeen is het slim als je ergens meer van wilt weten de help te bekijken. Behalve dat je daar vaak het antwoord op je vraag vindt, vind je er veel nuttige info.

Als we de meteor willen bewegen kunnen we natuurlijk de x en y-coördinaten gaan veranderen. Als je echter de meteor selecteert en kijkt in de rechterkolom bij de Properties kijkt zie je eigenschappen als Heading , Interval en Speed . Zet de Speed (inderdaad, de snelheid) van de meteor eens op 40 en kijk wat er gebeurt. Het Interval staat waarschijnlijk op 1000. Dat betekent dat elke 1000 milli-seconde (inderdaad, dat is 1 seconde) de meteor met zijn snelheid vooruit wordt gezet. Als de beweging schokkerig overkomt kun je het Interval kleiner maken. Maak er maar eens 200 van en kijk wat er gebeurt.

De beweging is minder schokkerig, maar ook sneller. Welke kant beweegt de meteor op?

#### Opgave 5.4

Met behulp van de Heading (dit is een hoek in graden) kun je aangeven welke kant de bal op moet bewegen. Probeer het uit of lees het in de help. Pas de waarde van Heading aan zodat de bal van boven naar beneden beweegt, dus in de richting van de raket.



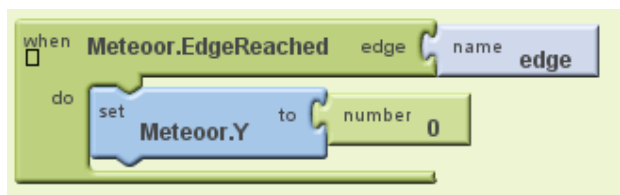
Als alles tot zover is gelukt zie je dat het meteorletje erg

klein is, maar als je de Radius op bijvoorbeeld 25 zet zie je dat de bal al heel wat groter is. Als het spel af is kun je uitproberen wat jouw favoriete grootte is en snelheid...

## 5.10 Meerdere meteoren

Als de meteor echter beneden is aangekomen blijft ie liggen. We willen eigenlijk dat ie dan weer van boven het scherm binnenkomt, wellicht in een andere grootte of kleur. Gelukkig helpt *App Inventor* ons ook hier weer uit de brand, namelijk met een event *EdgeReached*, dat optreedt als de bal (in het geval van `Meteor.EdgeReached`, wat te vinden is onder `Meteor` onder `My Blocks`).

Als de meteor beneden tegen de rand aan vliegt willen we dat de y-coördinaat weer 0 is, zodat de bal weer bovenaan begint. Als je `Ball1.EdgeReached` op het programmeerveld sleept zie je dat deze een parameter `name edge` heeft. Hiermee kun je kijken tegen welke kant de meteor gevlogen is. Aangezien wij de bal omhoog laten vliegen hoeven we niet te kijken, we weten al dat ie onder is aangekomen en boven opnieuw moet beginnen. In het block komt de code om de bal weer bovenaan het scherm te zetten.



**Figuur 5.9** Opnieuw boven beginnen

Het ziet er al bijna uit als een spel. De raket beweegt echter nogal schokkerig. Verder heeft een botsing geen gevolgen en tot slot blijft de meteor steeds op dezelfde plek naar beneden komen. Het schokkerige kunnen we een eind oplossen door de het Interval van de raket kleiner te maken, net als we bij de ball ook gedaan hebben. Klik op de raket en zet het interval op 100. Hoe kleiner het getal, hoe vaker de raket getekend wordt en dus hoe soepeler de beweging er uit ziet. Je zult echter wel snappen dat de mobiel het dan ook drukker heeft.



## 5.11 Botsing

Nu gaan we kijken naar het botsen. Ook hiervoor bestaat weer een event:



BlockEditor

|

My Blocks

|

Meteoor

|

Meteoor.CollidedWith

Het event treedt op als de Meteoor tegen een ander voorwerp botst en uit de parameter *other* kunnen we concluderen met welk voorwerp. Aangezien we al weten dat het de Raket is hoeven we dit niet te controleren: zo gauw het event optreedt weten we dat de Raket tegen een Meteoor geknald is. Voor nu zetten we dan de Meteoor stil, al is dat niet erg spectaculair. Sleep de code bij elkaar:



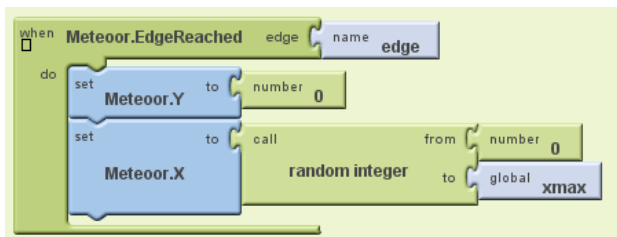
Figuur 5.10 Botsing



### Opgave 5.5

Hoewel in het echt in vacuüm natuurlijk geen geluid te horen is wordt het spel wel spectaculairder als je het geluid van een botsing toevoegt als de Raket tegen een Meteor botst.

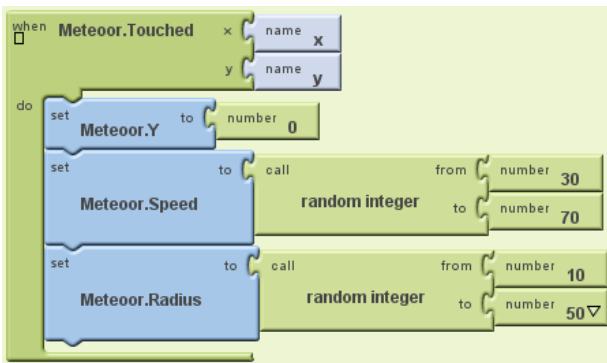
Als we de meteor succesvol ontwijken willen we dat ie op een willekeurige plek bovenin het scherm terugkomt, oftewel in het `EdgeReached` code-block willen we een willekeurige x-coördinaat zetten. `set Meteor.x to` , onder *Math* vind je een `random integer` -block. Hierin kun je *from* en een *to*-parameter invullen. Neem *From* = 0 en *To* = *xmax*, de eerder door ons berekende maximale x-coördinaat (zie *my blocks* | *my definitions*) die we hier handig kunnen hergebruiken.



Figuur 5.11 Botsing

## 5.12 En weer verder

Als de meteor stil staat en je klikt er op ( `Meteor.Touched` ) moet het spel weer verder gaan. Er moet een 'nieuwe' meteor komen van boven: hiertoe moeten we de y op 0 zetten en de snelheid (die we bij de botsing op 0 hebben gezet) weer op een goede waarde. We gebruiken een willekeurige (random) waarde tussen 30 en 70. Verder moeten de meteoren allemaal hun eigen grootte krijgen, ook *random* (willekeurig) dus.



**Figuur 5.12** Botsing

Het `Label` waar nog steeds de waarde van *roll* in gezet wordt kan inmiddels wel weg. Selecteer het in de `screen editor` en druk op `delete` (knop in kolom `Components` , rechterkant).



### 5.13 Mogelijke uitbreidingen

1. Iedere 'nieuwe' Meteoriet een willekeurige kleur.
2. Besturing andersom (en wellicht versnellend als mobiel scheefgehouden, ipv. constant)
3. Aantal levens
4. *Game over*: met een Label dat normaal onzichtbaar is en zichtbaar wordt gemaakt na een botsing kun je dit vrij eenvoudig maken.
5. Meerdere levels (bijvoorbeeld door de snelheid te verhogen, of misschien wel een tweede Meteor erbij)
6. Een ander plaatje als meteor? Je kunt hiervoor een `ImageSprite` gaan gebruiken.