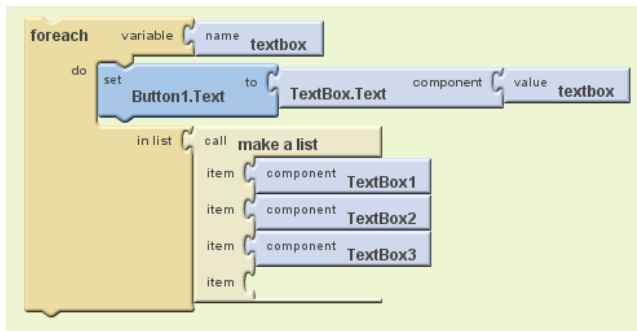


# Android Apps met App Inventor

Coen Crombach

Robin Eggenkamp

François Vonk



2 april 2016



Dit werk is gelicenseerd onder een Creative Commons Naamsvermelding-NietCommercieel-GelijkDelen 3.0 Unported. Bezoek <http://creativecommons.org/licenses/by-nc-sa/3.0/> om een kopie te zien van de licentie of stuur een brief naar Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

# 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.

**Belangrijk** In sommige gevallen willen we je iets belangrijks vertellen, bijvoorbeeld waarschuwing, een tip of een uitleg van een belangrijk begrip. Deze worden voorzien van een rode balk.

**Waarschuwing**

Hier staat een waarschuwing, lees deze zorgvuldig!



**Uitproberen!** Regelmatig kom je de afbeelding hiernaast 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>
1.1 Thuis installeren . . . . .	2
<b>2 Ontwikkelomgeving</b>	<b>5</b>
2.1 <i>Design</i> -scherm . . . . .	8
<b>3 Mollen Meppen</b>	<b>11</b>
3.1 Blocks Editor . . . . .	15
3.2 Willekeurig opduiken van de mol . . . . .	17
3.3 Geluid toevoegen als je de mol raakt . . . . .	19
3.4 De mol op een timer laten bewegen . . . . .	20
3.5 Tellen van het aantal maal dat je mis slaat . . . . .	22
3.6 Score toevoegen . . . . .	23
3.7 De mol sneller laten bewegen bij hoge score . . . . .	24
3.8 Testen op je telefoon . . . . .	25
3.9 Bonus opgaven . . . . .	26
<b>4 Schrandere Scholier</b>	<b>29</b>
4.1 Een nieuw project aanmaken . . . . .	29
4.2 Gebruikersinterface . . . . .	30
4.3 Rekenen . . . . .	32
4.4 Verbeteren . . . . .	35

4.5	Weer rekenen . . . . .	39
4.6	Bonus opgaven . . . . .	42
<b>5</b>	<b>Meteoor</b>	<b>45</b>
5.1	De Orientation Sensor . . . . .	46
5.2	De raket . . . . .	49
5.3	Wiskundig intermezzo . . . . .	51
5.4	Besturing . . . . .	53
5.5	xmax . . . . .	55
5.6	Geen Orientation Sensor? . . . . .	58
5.7	Raket onderaan het scherm . . . . .	60
5.8	Obstakels . . . . .	61
5.9	Meerdere meteoren . . . . .	63
5.10	Botsing . . . . .	64
5.11	Nog een stap verder . . . . .	66
5.12	Bonus opgaven . . . . .	67
<b>6</b>	<b>Project</b>	<b>69</b>
6.1	Brainstorm . . . . .	69
6.2	Ontwerpen . . . . .	70
6.3	Ontwikkelen . . . . .	71
6.4	Testen en verbeteren . . . . .	72
6.5	Presenteren . . . . .	73

# Hoofdstuk 1

## Installatie

De *App Inventor* ontwikkelomgeving draait in een webbrowser. 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 en een paar welkomstschermpjes weggeklikt hebt zie je een scherm zoals afgebeeld in figuur 1.1.



**Figuur 1.1** Het 'My Projects' scherm

Op school is alle software die *App Inventor* nodig heeft al voor je geïnstalleerd. Als je echter 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.

## 1.1 Thuis installeren

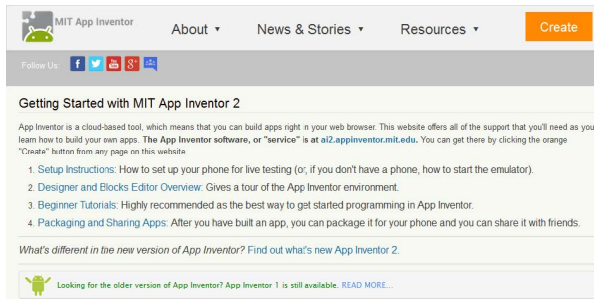
Het is nu belangrijk dat je de software installeert die *App Inventor* nodig heeft om de *Blocks Editor* en de emulator uit te voeren. Deze termen zullen je nu misschien nog niets zeggen en dat is op dit moment niet erg. Verderop in het materiaal, als je ze nodig hebt, zullen ze in detail behandeld worden.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



**Figuur 1.2** de 'Get Started' link

Als je dit hebt gedaan zie je in je browser de pagina 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** Getting Started webpagina

Vervolgens klik je op link 1: 'Setup Instructions' en volgt de instructies die je krijgt via de links bij 'Option Two' en 'Option Three'.



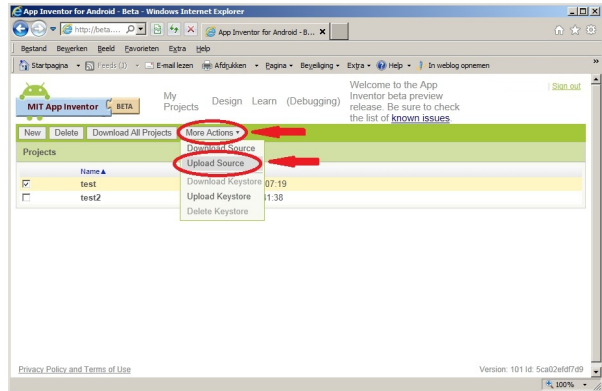
## Hoofdstuk 2

# Ontwikkelomgeving

### Waarschuwing

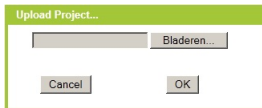
Voordat we van start gaan met *App Inventor* een kleine waarschuwing. De ontwikkelomgeving is niet altijd even snel en het gaat niet sneller door veelvuldig op links of knoppen te klikken. Sterker nog: dit kan er voor zorgen dat de omgeving nog langzamer wordt en dat er fouten op gaan treden. Om dit 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.



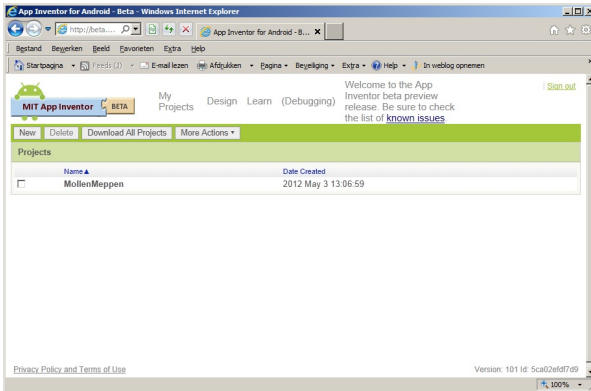
**Figuur 2.1** Locatie van 'Upload Source'

In ons geval gaan we het 'MollenMeppen' project binnenhalen. App Inventor projecten worden opgeslagen als ZIP bestanden en het is de bedoeling dat je het ZIP bestand *uploadt*. Pak het bestand dus NIET uit want dan werkt het *uploaden* niet! 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 (ZIP bestand) dat we willen binnenhalen. Als je niet weet waar je het 'MollenMeppen' project kunt vinden vraag het dan aan je docent. Als we het projectbestand hebben gevonden dan selecteren we het en klikken vervolgens op **OK**.



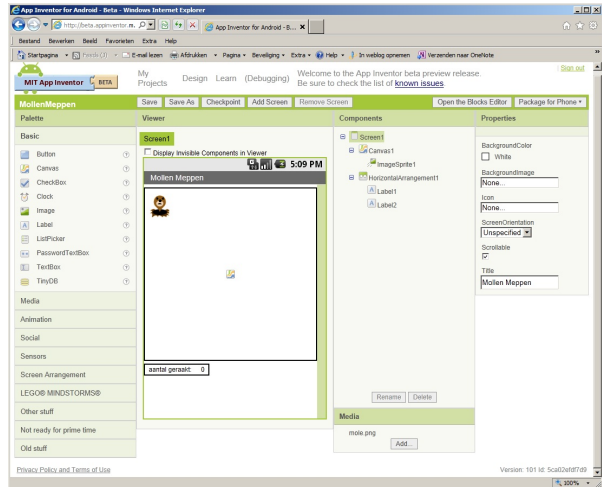
**Figuur 2.2** 'Upload Project' pop-up

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 MollenMep-  
pen project

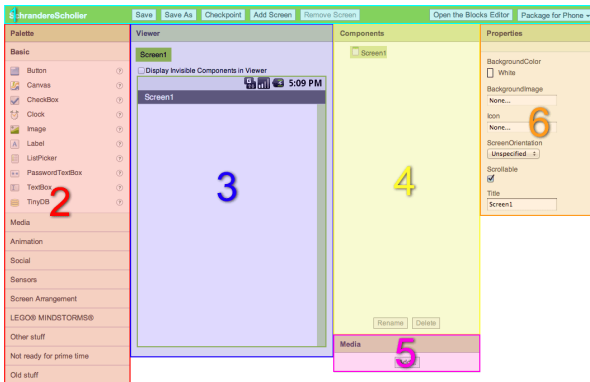
Korte tijd nadat het project is toegevoegd zal de ontwik-  
kelomgeving automatisch naar het *Design*-scherm sprin-  
gen waarin we het ontwerp van de applicatie zien, zie figuur  
2.4.



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

## 2.1 *Design*-scherm

Het *design*-scherm bestaat uit verschillende onderdelen, om straks de gebruikersinterface van je eigen applicatie te kunnen ontwerpen is het belangrijk de verschillende onderdelen hiervan te kennen. Deze onderdelen zie je in figuur 2.5.



**Figuur 2.5** 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.

Om weer naar het 'My Projects' scherm te gaan kun je deze aanklikken in de ontwikkelomgeving.

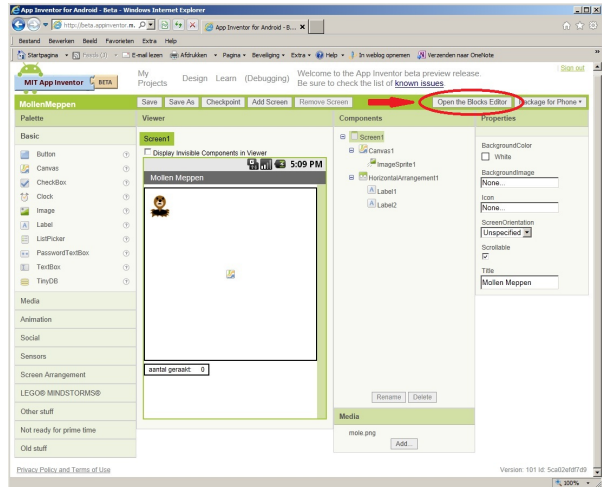


## Hoofdstuk 3

# Mollen Meppen

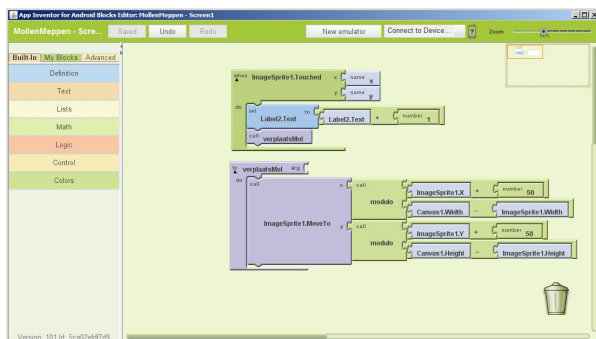
Je hebt zojuist je eerste project geladen en het ontwerp ervan kort gezien. Voor je het een en ander uitgebreid bekijkt ga je de applicatie eerst een keer uitvoeren.

Dit doe je door vanaf het **Design** scherm 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. Tot slot wordt de **Blocks Editor** geopend en zie je wat er in figuur 3.2 is afgebeeld.

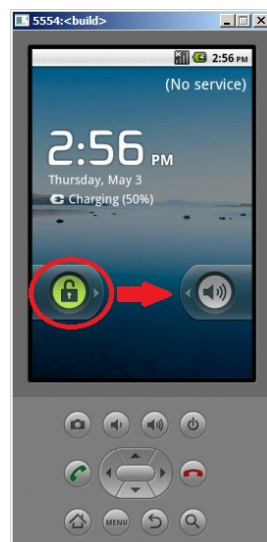


**Figuur 3.2** Blocks Editor van MollenMeppen

### Let op

De **Blocks Editor** is een aparte applicatie die niet in je webbrowser draait. Je ziet in je taakbalk dan ook een *App Inventor for Android Blocks Editor: MollenMeppen - Screen1* pictogram. Je kunt met 'ALT-TAB' makkelijk wisselen tussen het **Design** scherm en de **Blocks Editor**.

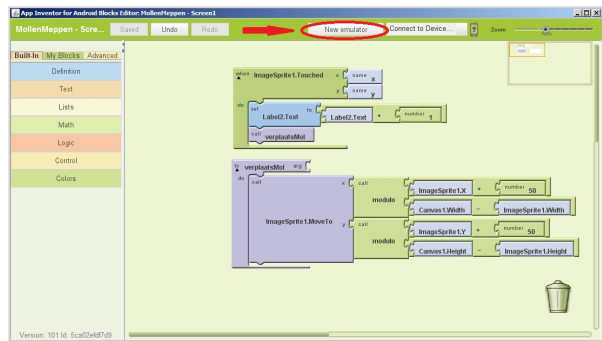
Om de applicatie uit te voeren moet je eerst een emulator opstarten. Dit doe je via de **New emulator** link zoals aangegeven in figuur 3.4. Als de emulator is opgestart ziet deze er uit zoals afgebeeld in figuur 3.3. De emulator staat nu nog 'op slot', je kunt het slot verwijderen door met de muis op 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 3.3** De emulator

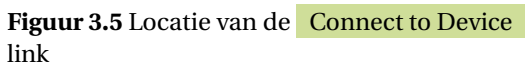
### Let op

Zorg ervoor dat je de emulator maar één keer opstart. Het meerdere keren opstarten van de emulator leidt tot een instabiele situatie en zorgt ervoor dat op den duur de boel kan crashen.



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

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




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 plezier en succes!

### 3.1 Blocks Editor

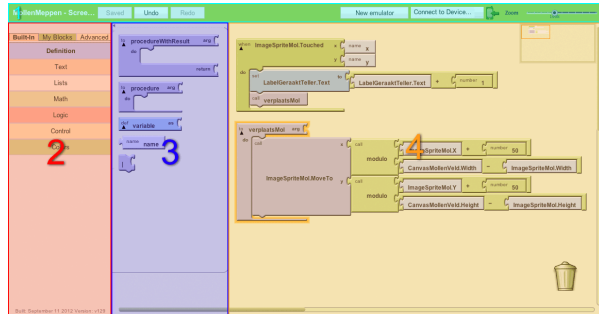
5:55:<build>

Mollen Meppen



santal geraakt:0

**Figuur 3.6** De Mollen-Meppen applicatie



**Figuur 3.7** De verschillende onderdelen van de Blocks Editor

1. **De knoppenbalk** In deze balk vind je knoppen om het project op te slaan, stappen ongedaan te maken en om met de emulator te communiceren.
2. **Palette** Het palette is de plek waar je alle categorieën van *blocks* vindt die je kunt gebruiken in jouw apps. Onder de tab **My Blocks** vind je de door jouw aangemaakte componenten. Hierachter vind je de bijbehorende *blocks*, hiermee kun je bijvoorbeeld een actie koppelen aan een knop.
3. **Blocks** Als je in het palette een categorie aanklikt verschijnt dit overzicht. Hierin vind je de *blocks* zelf die je naar het canvas kunt slepen om ze te gebruiken.
4. **Canvas** In dit deel ontwerp je de gebruikersinterface van je applicatie. Vanuit het palette sleep je *blocks* in het canvas. Wil je een *block* verwijderen, dan sleep je deze naar de prullenbak rechtsonder.

**Let op**

Op het moment dat je een component in het Design scherm sleept of een bestaande component een andere naam geeft, dan moet de Blocks Editor automatisch bijgewerkt worden. Helaas gaat dit niet altijd goed. Als je merkt dat de Blocks Editor niet meer klopt bij je Design scherm, sluit de Blocks Editor dan af en start hem opnieuw op.

## 3.2 Willekeurig opduiken van de mol

De mol beweegt zich erg voorspelbaar op dit moment en dat maakt het spel saai. In de Blocks Editor kun je 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.

**Procedures**

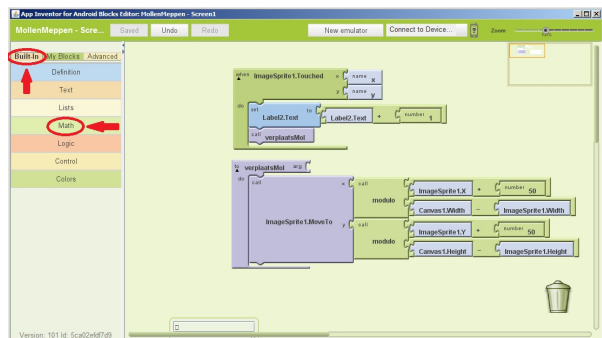
Als je de code bekeken hebt zoals in de opgave hiervoor gevraagd werd, is je hopelijk opgevallen dat er een zogenaamde procedure is gemaakt met de naam ‘verplaatsMol’. Deze procedure wordt aangeroepen vanuit ‘ImageSpriteMol.Touched’.

Een procedure (in veel programmeertalen ook wel *functie* genoemd) is een stuk code met een naam. In

ons geval 'verplaatsMol'. Via de naam kan de bijbehorende code uitgevoerd worden. De naam is als het ware een afkorting voor het stuk code.

Procedures zijn handig omdat ze voorkomen dat je hetzelfde stuk code steeds opnieuw moet maken als je die code op meerdere plaatsen nodig hebt. Dit is minder werk en beperkt tevens de lengte van je applicatie. Wanneer je bovendien duidelijke namen gebruikt is je code ook makkelijker te lezen. Dit alles draagt bij aan de overzichtelijkheid en daarmee de onderhoudbaarheid van de code.

Wat je hier feitelijk wilt is dat de  $x$  en  $y$  positie van de mol (de *ImageSprite*) willekeurig bepaald worden. Hiervoor kun je een *block* vinden onder de **Built-In** tab bij **Math**, zie figuur 3.8. *Blocks* uit de **Blocks Editor** kun je simpelweg in het **canvas** 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 3.8** Locatie van de 'Built-In' Math link



**Opgave 3.2**

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

👉 Hint: Wat is het Engelse woord voor willekeurig?

## 3.3 Geluid toevoegen als je de mol raakt

Als de mol geraakt wordt is het leuk als je dat ook hoort. Gelukkig kun je dit programmeren in *App Inventor*. Bij het materiaal zit een bestand *whack.mp3* dat je hiervoor kunt gebruiken. Als je dit bestand niet kunt vinden vraag dan aan je leraar waar het staan.

Om *whack.mp3* in je project te krijgen moet je het geluidsbestand toevoegen aan de media. Dit kun je doen via het **Design** scherm. Als je niet kunt vinden waar je moet zijn kijk dan eens terug in dit boek naar figuur 2.5.

**Opgave 3.3**

Voeg het geluid toe aan het spel.

Als je *whack.mp3* hebt toegevoegd en je gaat naar de **Blocks Editor** dan zie je dat onder de **My Blocks** tab **Sound1** is toegevoegd. Hieraan zijn diverse *blocks* gekoppeld die met het afspelen van geluiden te maken hebben.

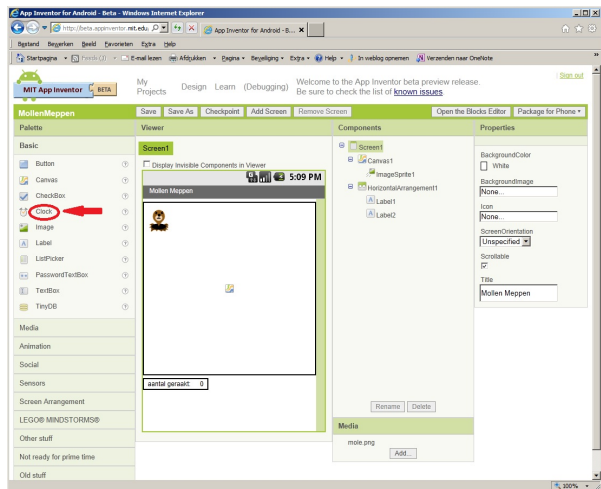
**Opgave 3.4**

Speel het toegevoegde geluid af op het moment dat de mol geraakt wordt.

### 3.4 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 ga je een zogenaamde **timer** gebruiken.

Een timer kun je vinden via de **Clock** component in het **Basic** palette, zie figuur 3.9. 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.



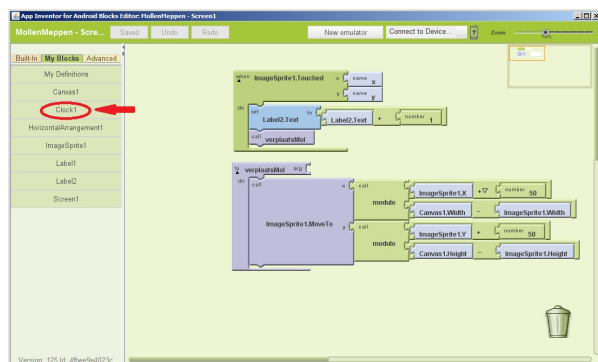
**Figuur 3.9** Locatie van de **Clock** component in het **Basic** palette

### Opgave 3.5

Voeg een **Clock** component aan het spel toe en kijk goed waar deze neergezet wordt. Staat daar ook nog een andere component?

Beredeneer waarom de clock en de andere component hier neergezet worden.

Nadat je de **Clock** component hebt toegevoegd kun je de timer programmeren. Hiervoor moet je 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 **Clock1** link omdat je de **Clock** component hebt toegevoegd, zie figuur 3.10. Klik op de link en je ziet bovenaan een **Clock1.Timer** block.



**Figuur 3.10** Locatie van de **My Blocks** | **Clock1** link

### Opgave 3.6

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

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

Als je niet weet hoe een timer werkt of je komt er niet helemaal uit, klik dan eens op de 'Learn' link in je **Design** scherm. Je kunt dan bij de *Reference Documentation* op zoek gaan naar hoe een timer werkt.

Hopelijk valt je op dat de documentatie net zo is opgezet als de ontwikkelomgeving van *App Inventor* zelf. Er wordt onderscheid gemaakt tussen componenten (dingen uit je **Design** scherm) en blocks (dingen uit de **Blocks Editor**).

Kijk eens goed in je **Design** scherm waar de **Clock** component zit en ga dan in de documentatie zoeken.

### Procedures

Als het goed is heb je nu voordeel gehad van de procedure die we al voor je gemaakt hebben. In plaats van de code opnieuw te maken heb je als het goed is nu de procedure aangeroepen.

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

Naast het bijhouden hoe vaak een speler de mool raakt wil je ook graag bijhouden hoe vaak de speler mis slaat. In je **Components** staat een **HorizontalArrangementGeraakt** die ervoor zorgt dat je componenten horizontaal in je scherm kunt rangschikken. De componenten erin (**LabelGeraaktTekst** en **LabelGeraaktTeller**) komen overeen met 'aantal geraakt' en '0' in je **Viewer**.

**Let op**

Een label gebruik je om een stuk tekst weer te geven. Een TextBox gebruik je om de gebruiker iets in te laten voeren.

**Opgave 3.7**

Breid de bestaande interface uit met een `HorizontalArrangement` dat daarin labels heeft voor '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.

 Hint: Op het moment dat je de mol raakt gaan er twee events af namelijk 'CanvasMollenVeld.Touched' en 'ImageSpriteMol.Touched'. Hoe kun je deze slim gebruiken om te bepalen wanneer je mis slaat?

## 3.6 Score toevoegen

Het bijhouden van het aantal keer raak en mis slaan is leuk maar een score is nog leuker. Voor iedere keer dat een speler de mol raakt krijgt hij/zij 2 punten. Voor iedere keer dat de speler de mol mist verliest hij/zij een punt.

**Opgave 3.8**


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.

### 3.7 De 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 dus helaas niet steeds meer van het reactievermogen van de speler maar dat kun je veranderen.

Daarvoor moet je onder bepaalde voorwaarden de mol steeds sneller laten bewegen. Om dit voor elkaar te krijgen heb je een zogenaamd *if-statement* nodig. Zo'n *if-statement* kun je vinden in de **Blocks Editor** in de tab **Built-In** onder **Control**. Daar vindt je de **if** en **ifelse** blocks.

Bij een *if-statement* wordt er getest of een expressie waar of niet waar is (we noemen zo'n expressie een *Booleaanse expressie*). Als de expressie waar is wordt de code die binnen het *if-statement* staat wel uitgevoerd en anders wordt deze code overgeslagen.

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

#### Opgave 3.9

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.

#### Conditioneel uitvoeren van code

Het conditioneel uitvoeren van code gebeurt bij het programmeren vaak met *if-statements*. Er zijn echter ook nog andere manieren zoals bij *App Inventor* het **choose** block.

*if-statements* zijn belangrijk en essentieel bij het programmeren. Ze zorgen ervoor dat delen van het pro-

gramma alleen onder bepaalde voorwaarden worden uitgevoerd.

We noemen dit program control en het geeft gebruikers van onze applicatie de mogelijkheid om de flow van het programma te beïnvloeden.

## 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 versie 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**.

### 3.9 Bonus opgaven

 **Tip:** Kijk eens bij 'Properties' van de **canvas** component.


#### Opgave 3.10

De achtergrond van het spel is op het moment saai wit. Voeg een achtergrond toe aan het spel om de boel wat op te fleuren.

#### Opgave 3.11

Je kunt op dit moment nog niet afgaan bij het spel. Verander dit door de speler af te laten gaan wanneer zijn/haar score onder de 0 punten zakt.

Het afgaan in het spel kun je ook op een heel andere manier aanpakken. Je kunt de speler bij het begin van het spel een aantal levens geven en onder bepaalde omstandigheden laten verliezen. De speler is af wanneer hij/zij geen levens meer over heeft.

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



#### **Opgave 3.12**

Bouw het concept van levens in je spel in. Om levens te verliezen voeg je nog een ander dier toe aan het spel, bijvoorbeeld een lief konijntje of juist een gruwelijk monster. Laat het andere dier af en toe verschijnen in plaats van de mol en wanneer de speler het andere dier mept verliest hij/zij een leven.



# Hoofdstuk 4

## Schrandere Scholier

In het vorige hoofdstuk heb je een leuk spel gemaakt, je kunt 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 ga je hier een hulpmiddel voor ontwikkelen.

Je zult de applicatie stap-voor-stap opbouwen. Je maakt een begin met de gebruikersinterface in het **Design**-scherm.

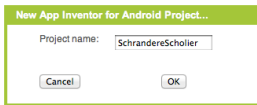
Daarna zul je in de **Blocks**-editor ervoor zorgen dat de gebruiker cijfers kan invoeren waarmee je vervolgens kunt gaan rekenen. Tot slot zorg je ervoor dat de gegevens opgeslagen worden zodat je niet bij elke start opnieuw hoeft te beginnen.

### 4.1 Een nieuw project aanmaken

Je begint met het aanmaken van een nieuw project. In hoofdstuk 2 heb je een project aangemaakt door een be-

staand project te uploaden. Dit keer begin je 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

Na het aanmaken van het project begin je met het ontwerpen van de gebruikersinterface. Voordat je hiermee aan de slag gaat: bekijk de verschillende onderdelen van het **Design**-scherm nog eens in paragraaf 2.1.

Nu je een idee hebt van de interface kunnen we deze gaan gebruiken. Je begint 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'.

- ☞ Een label gebruik je om een stukje tekst weer te geven.



**Figuur 4.2** Label

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

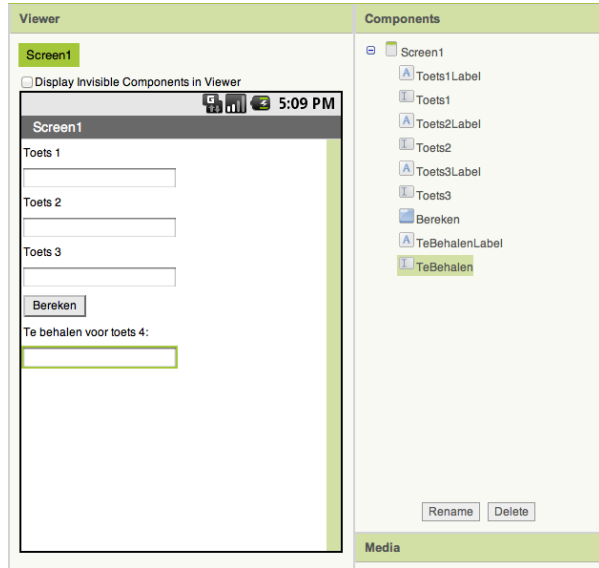
### Opgave 4.1

Herhaal deze handelingen totdat je drie toetsen kunt invoeren.

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 doe je later, maar je voegt nu al wel een knop en twee labels toe om het resultaat weer te geven. Een knop voeg je toe door een **Button** vanuit het Palette naar de Vierwer 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 4.2

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 ga je nu iets aan doen!

## 4.3 Rekenen

 Een uitgebreidere uitleg vind je in hoofdstuk 2

Je gaat nu zorgen dat de applicatie daadwerkelijk kan rekenen. Hiervoor moet je programmeren en dit doe je 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 codeblokken 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 codeblokken vinden.

Je wilt 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 codeblokken die je in dit block hangt worden uitgevoerd als er op de knop geklikt wordt.

Je begint met het verzamelen van de gegevens waarmee je straks kunt rekenen. De cijfers moet je 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**. Je hebt dus nog andere blocks nodig.

Je wilt straks kunnen 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 noem je een *expressie*. In **Bereken.Click** passen enkel blocks die een *actie* vertegenwoordigen, dit wordt ook wel een *statement* genoemd.

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.

### Opgave 4.3

Na het rekenen zul je 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 je verder gaat: probeer uit of het programma werkt zoals je verwacht. Als het werkt dan wordt het tijd om jouw programma uit te breiden. Er vanuit gaande dat alle cijfers dezelfde weging hebben, hoe kun je dan eenvoudig 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. Iemand heeft bijvoorbeeld een 9 en tweemaal een 7 gehaald. Dit is in totaal  $9 + 7 + 7 = 23$ . Het in dit geval te behalen cijfer is dus  $32 - 23 = 9$ .

### Opgave 4.4

Implementeer de bovenstaande berekening en zet het antwoord in `TeBehalen`. Je hebt hiervoor,



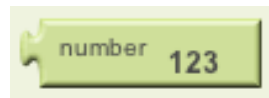
naast de al bekende blocks, ook het aftrek-block nodig en een constant getal. Een constant getal geef je aan met het block in figuur 4.6.

Je moet verschillende blocks 'in elkaar hangen', dit wordt *nesten* genoemd. Bij het uitrekenen kijkt de computer welke gegevens hij nodig heeft om het block uit te kunnen rekenen, deze gegevens zal hij eerst opzoeken of berekenen. Zodra de computer alle gegevens heeft om een block uit te rekenen doet hij dit. In het voorbeeld van een optel-block zal hij dus eerst bepalen welke twee getallen opgeteld moeten worden, om deze vervolgens op te tellen en door te geven.

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



**Figuur 4.7** Uitwerking



**Figuur 4.6** Constant getal

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 zul je deze applicaties verder gaan uitwerken. Je zult daarbij meer zelfstandig aan de slag gaan met *App Inventor*.



## 4.4 Verbeteren

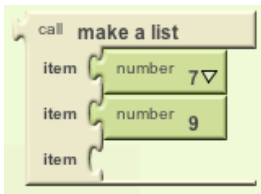
De eerste verbetering die je gaat doorvoeren is het toevoegen van meer cijfers, waarschijnlijk heb je geen enkel vak

waarbij je maar vier toetsen krijgt.

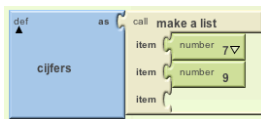
### Opgave 4.5

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. Zou je dit kunnen automatiseren? Dan kun je 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. Je kunt 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 ga je in deze sectie implementeren.



**Figuur 4.8** make a list block



**Figuur 4.9** De lijst in een variabele

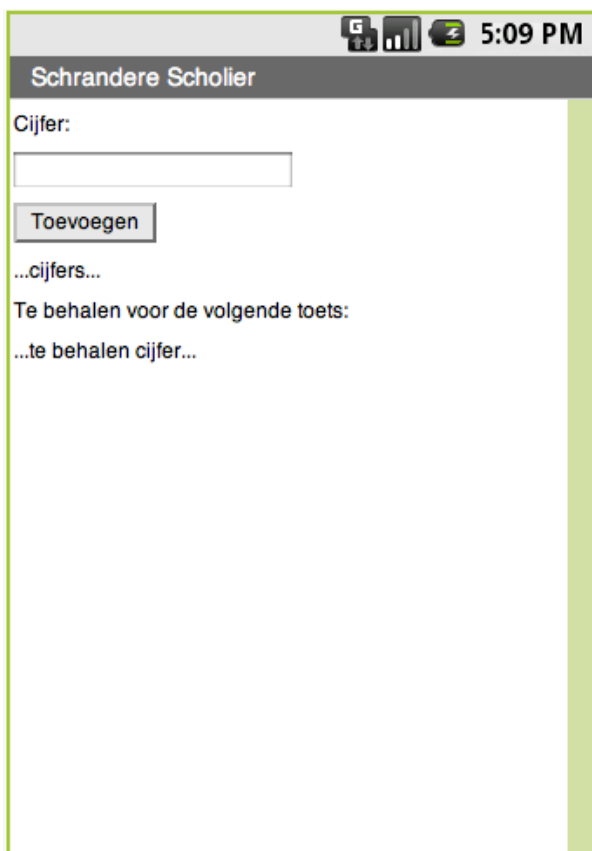
Allereerst maak je in de **Blocks Editor** de lijst aan. Je doet dit door het **make a list** block (uit het **Lists** palette) het werkblad op te slepen. Zoals je ziet kun je in dit block items toevoegen, dit kan bijvoorbeeld een **number** zijn. Een voorbeeld zie je in figuur 4.8.

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

### Opgave 4.6

De interface kun je nu aanpassen. Maak de interface


zoals is weergegeven in figuur 4.10. Er staan twee labels in het ontwerp waarvan je later de tekst gaat aanpassen vanuit de **Blocks editor**. Denk ook nu aan een duidelijke naamgeving van de componenten.



**Figuur 4.10** Nieuwe gebruikersinterface

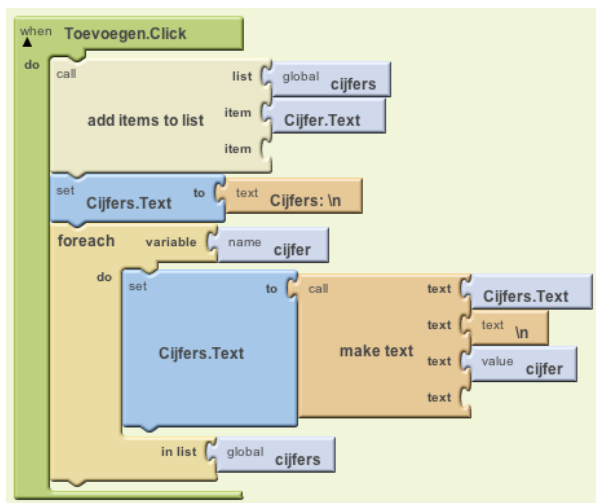
Ga nu weer naar de **Blocks editor**. Als je de knop enkel hebt hernoemd heb je hier nog een block **Toevoegen.Click** staan. Indien dit niet het geval is voeg je deze toe vanuit **My Blocks**. In deze procedure moet je het nieuw ingevoerde cijfer toevoegen aan de lijst. Een item toevoegen aan de lijst 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 moet je een 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 kun je dit gebruiken om de cijfers in de list allemaal onder elkaar te zetten. Hoe je dit doet zie je in figuur 4.11.

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

#### Opgave 4.7

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



**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

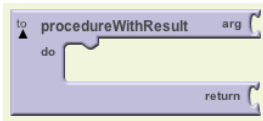
De oude methode om het benodigde cijfer te berekenen werkt nu niet meer, daar moet je iets aan doen. Omdat alle cijfers in een lijst staan moet je hier weer een **loop** block voor gebruiken. Om te voorkomen dat de code onoverzichtelijk wordt zul je de berekening in een aparte *procedure* laten plaatsvinden.

### Procedures met resultaat

Op pagina 17 staat uitgelegd wat een procedure is. In het kort is een procedure 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. Dit pas je toe om bijvoorbeeld een berekening die je meerdere malen uitvoert maar eenmalig te hoeven op te schrijven.

Ook kan het handig zijn om een procedure te gebruiken om je code leesbaarder te maken. Een lang stuk code kun je zo opdelen in kleine, begrijpelijke stukjes code. Zorg er altijd voor dat je de procedure een betekenisvolle naam geeft.



**Figuur 4.12** procedureWithResult block

Een nieuwe procedure met resultaat kun je aanmaken door het `procedureWithResult` block, te vinden in het

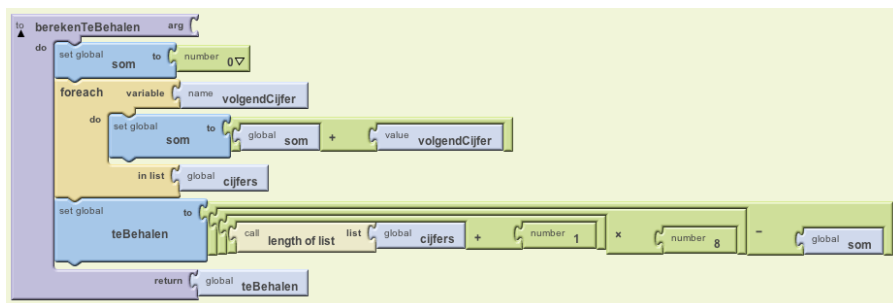
**Definition** palette, naar het canvas te slepen. Je kunt er op drie plekken blocks aanhangen. Allereerst bij 'arg', hier kun je een argument meegeven aan de procedure, dit zul je hier nog niet gebruiken. Bij 'do' voeg je de acties toe die moeten gebeuren. Bij 'return' geef je een waarde die wordt teruggegeven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

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

Je begint met het optellen van alle resultaten. Dit kun je doen op een vergelijkbare manier als het aan elkaar plakken van de teksten, zoals je dat gedaan hebt in figuur 4.11. Je gebruikt nu echter hetzelfde `+` block als je eerder gebruikt hebt. 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



**Figuur 4.13** som variabele



**Figuur 4.14** Implementatie van berekenTeBehalen

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.

### Opgave 4.8

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.

### Opgave 4.9

Voeg de blocks om het te behalen cijfer weer te geven 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 dan heb je in de Blocks Editor nog invulling staan bij het aanmaken van de lijst. Haal deze weg en probeer het opnieuw!

**Opgave 4.10**

De implementatie van de procedure `berekenTeBehalen` is al behoorlijk ingewikkeld! Probeer de code leesbaarder te maken door de procedure op te delen in enkele korte procedures. Denk steeds goed na over welke statements je opneemt en welke naam je aan de procedure geeft.

## 4.6 Bonus opgaven

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.

**Opgave 4.11**

Een 8 gemiddeld, waarom geen 9? Laat de gebruiker zelf kiezen welk cijfer hij gemiddeld wil staan.

**Opgave 4.12**

Niet alle toetsen tellen even zwaar, zorg ervoor dat de gebruiker het gewicht van een toets kan bepalen.



### **Opgave 4.13**

De app ziet er op dit moment saai uit, zorg ervoor dat de app er aantrekkelijk om te gebruiken uitziet.

### **Opgave 4.14**

Elke keer als je de app opnieuw opstart ben je je cijfers kwijt. Zou het niet mooi zijn als de cijfers bewaard blijven? Zorg ervoor dat de cijfers worden bewaard bij het opnieuw opstarten van de app.

### **Opgave 4.15**

Je kunt nu voor één vak tegelijk cijfers invoeren, maak de app geschikt voor meerdere vakken.

### **Opgave 4.16**

Heb je geprobeerd om het cijfer  $-3$  in te voeren, of het cijfer  $C$ ? Je zult zien dat dit kan! Maak het onmogelijk om foutieve cijfers in te voeren.

### **Opgave 4.17**

Heb je zelf een voorstel tot verbetering? Overleg dit met je docent, en implementeer het!



# Hoofdstuk 5

## Meteoor

In dit hoofdstuk ga je zelf een eenvoudig spel maken. Je gaat hiervoor de *Orientation Sensor* van je mobiel gebruiken. Deze sensor geeft aan of het apparaat gekanteld wordt of niet. We helpen je in dit hoofdstuk om het spel stap voor stap op te bouwen.

Je zult merken dat je in dit hoofdstuk minder gedetailleerde aanwijzingen krijgt en dus meer zelfstandig moet doen. Kom je ergens niet uit, blader dan terug naar één van de voorgaande hoofdstukken en als je er dan nog niet uitkomt vraag het aan je docent.

Allereerst ga je uitzoeken hoe de *Orientation Sensor* werkt. Je gaat experimenteren door het aansturen van een `ImageSprite` (zoals de mol in het ‘Mollen Meppen’ spel). In dit spel is de `ImageSprite` een raket. Je bestuurt de raket door je telefoon naar links of rechts te kantelen.

Je krijgt ook aanwijzingen over hoe je het spel kunt maken zonder de *Orientation Sensor*. Dit kan nodig zijn als je

mobiel geen sensor heeft of als je aangewezen bent op de emulator van *App Inventor*. Als je door hebt hoe de Orientation Sensor werkt ga je meteorieten over het scherm laten bewegen. Vervolgens ga je detecteren of de raket geraakt wordt door een meteoriet om te bepalen wanneer je af bent.

## 5.1 De Orientation Sensor

### Opgave 5.1

Maak allereerst een nieuw project in *App Inventor* en noem het ‘meteor’.

Nadat je een project hebt aangemaakt ga je zichtbaar maken wat de toestand van de Orientation Sensor is. Daarvoor gebruik je labels in het Design-scherm.

In het Sensors palette vind je de OrientationSensor. Sleep ook deze in het scherm. Net als de Clock component bij ‘Mollen Meppen’ komt de Orientation Sensor ook onder in het scherm te staan bij de *Non-visible components*. Tijdens het uitvoeren van het programma is deze sensor namelijk niet zichtbaar, maar tijdens het ontwikkelen wil je natuurlijk wel ergens aan kunnen zien dat de sensor er staat.

### Opgave 5.2

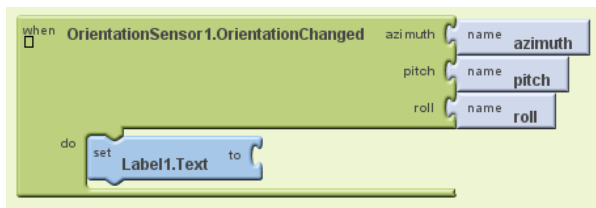
Sleep drie Labels uit het Basic palette in het scherm. Het is een goed gebruik om de componenten in je programma meteen een duidelijke naam te geven. Sleep ook een OrientationSensor in het scherm.

Vervolgens ga je programmeren in de **Blocks-Editor**. De blocks van de Orientation Sensor vind je via de tab **My Blocks**. Als je op de **Orientation Sensor** component klikt zie je onder andere het **OrientationSensor1.OrientationChanged** block. Dit is een event (een gebeurtenis net als de *timer* die je bij ‘Mollen Meppen’ hebt gebruikt) dat optreedt als je je mobieltje kantelt. Als dit event optreedt, wordt de code die je in het **OrientationSensor1.OrientationChanged** block zet uitgevoerd.

### Opgave 5.3

Voer de volgende opdrachten uit:

- Sleep het **OrientationSensor1.OrientationChanged** block in het programma gedeelte. Bovenaan in het block staan aan de rechterkant drie parameters genaamd *azimuth*, *pitch* en *roll*.
- Klik in de tab **My Blocks** op de naam van het eerste label en sleep dan het **set Label1.Text to** block in het **OrientationSensor1.OrientationChanged** block zoals te zien is in figuur 5.1.



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

### Parameters en argumenten

Een parameter is een speciale variabele die je gebruikt om informatie door te geven aan een procedure. Parameters staan bovenaan in het block aan de rechterkant. Een procedure kan meer dan één parameter hebben.

Een argument is een waarde die je meegeeft aan een procedure.

Procedures zijn uitgelegd bij Mollen Meppen.

Daarna ga je via het `set Label1.Text to` block de waarde van de eerste parameter (*azimuth*) laten zien. Deze is bereikbaar via de **My Blocks** tab onder **My Definitions**.

### Opgave 5.4

Klik `value azimuth` vast aan het `set Label1.Text to` block.

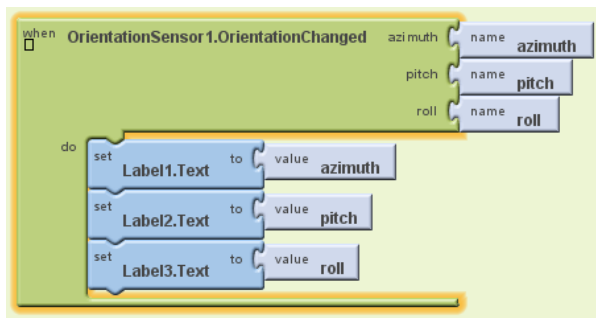
Laat vervolgens ook de waarde van de tweede en derde parameter (*pitch* en *roll*) zien.

Als je code eruit ziet zoals afgebeeld in figuur 5.2 ben je klaar om te gaan testen.



Zet de app op je mobiel en start het op. Kantel het apparaat naar links en naar rechts en kijk naar de getallen die in de app afgebeeld worden. De waarde van *roll* is 0 als je het toestel waterpas houdt. Als je het toestel naar links kantelt geeft de *roll* aan hoeveel graden het apparaat gekanteld is (90 graden als je het toestel op zijn kant houdt). Als je het toestel naar rechts kantelt worden negatieve graden aangegeven (–90 graden als je het toestel op zijn kant houdt).

Met wat fantasie kun je de app die je nu hebt al gebruiken als een soort waterpas.



Figuur 5.2 waarden

### Opgave 5.5

Probeer zelf uit te vinden wat de andere twee getallen aangeven.

Hint: Als je er niet uitkomt kun je de woorden *azimuth* en *pitch* opzoeken op het internet.

## 5.2 De raket

In deze paragraaf ga je de raket maken en hem besturen met de Orientation Sensor via de *roll* parameter. De labels *pitch* en *azimuth* in het Design-scherm kun je weggooien, omdat je ze niet nodig hebt bij het spel.

### Opgave 5.6

Selecteer de labels één voor één en klik daarna op de button **Delete**.

Als je met ALT+TAB naar de **Blocks-Editor** gaat zie je dat de bijbehorende blocks ook verdwenen zijn.

In het **Animation** palette vind je de **ImageSprite** component. Deze ga je gebruiken om de raket te maken. Een *sprite* moet in een *canvas* geplaatst worden om deze zichtbaar te maken en goed te laten functioneren. De **Canvas** component kun je vinden in het **Basic** palette.

### Opgave 5.7

Sleep het **Canvas** in het scherm en sleep de **ImageSprite** daar weer in. Zet de sprite ongeveer midden in het canvas. Klik vervolgens op de **Canvas** component zodat deze geselecteerd is.

In het **Properties** gedeelte van het **Design** -scherm zie je de eigenschappen van het canvas. De bovenste eigenschap is de **BackgroundColor** (achtergrondkleur). Via deze eigenschap kun je een andere achtergrondkleur kiezen.

### Opgave 5.8

Kies een andere achtergrondkleur voor het canvas. Als je dat gedaan hebt experimenteer ook met de andere eigenschappen van het canvas.

De meest interessante eigenschappen zijn op dit moment de onderste twee namelijk *Width* en *Height*.

### Opgave 5.9

Stel beide eigenschappen in op '*Fill parent...*'. Probeer te verklaren wat je ziet gebeuren.

Als het goed is zie je dat de breedte van het canvas even breed wordt als het scherm. Bij de hoogte is dit echter niet het geval omdat het scherm kan scrollen.



**Opgave 5.10**

Selecteer onder **Components** in het **Design** - scherm de **Screen** component. Zet de eigenschap *Scrollable* uit en kijk wat er gebeurt. Als het goed is zie je de hoogte van het *Canvas* even hoog worden als het scherm.

Nadat je dit allemaal gedaan hebt ben je klaar om van de sprite een raket te maken. Dit kun je doen via **Media** in het **Design** scherm, net zoals je al geluid hebt toegevoegd bij 'Mollen Meppen'. Pak een plaatje van een raket dat je bij het materiaal krijgt, bijvoorbeeld 'raket2.jpg'. Als je geen plaatje van een raket kunt vinden vraag er dan één aan je docent. Als je eenmaal klaar bent met het spel kun je zelf een ander (niet te groot) plaatje ervoor in de plaats zetten. Experimenteer en leer!

**Opgave 5.11**

Voeg het plaatje van de raket toe aan je project. Kop-pel vervolgens het plaatje aan de sprite. Op het canvas kun je nu je raket bewonderen.

## 5.3 Wiskundig intermezzo

**Noot**

Je kunt het spel ook maken zonder de wiskunde achter de Orientation Sensor precies te snappen. Maar als je geïnteresseerd bent in de wiskunde bekijk deze paragraaf dan goed. Deze paragraaf mag je overslaan als je wilt.

Laten we zeggen dat als je 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 ( $Canvas.Width$ ). Je moet daar echter de breedte van de raket nog vanaf halen. Ofwel: de maximale  $x$ -coördinaat is

$$Canvas.Width - ImageSprite.Width$$

Aangezien de waarde van deze expressie constant is sla je deze op in een variabele, die je  $xmax$  noemt. Je hebt 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*: je hebt twee waarden voor  $roll$  en wilt een berekening (lees: functie) die de bijbehorende  $x$ -coördinaat uitrekent. Bij wiskunde zou je dit opschrijven als:

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

 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.

Raak niet in de war komt doordat hier de  $x$  aan de linkerkant staat! Je kunt 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 één geleerd. Aangezien  $-45$  en  $45$  even ver aan beide kanten van de  $y$ -as liggen weet je 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 kun je de waarde van  $a$  berekenen.

### Opgave 5.12

Laat zien dat hier uitkomt:

$$a = xmax/90$$

Je kunt nu  $f(x)$  afleiden:

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

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:

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

## 5.4 Besturing

### Noot

Als je geen Orientation Sensor op je mobiel of geen mobiel hebt ga dan naar paragraaf 5.6: 'Geen Orientation Sensor?'.

Alle voorbereidingen zijn klaar. Nu ga je bepalen wat er moet gebeuren als je mobiel wordt gekanteld; dus als het `OrientationSensor1.OrientationChanged` event optreedt. Hiervoor ga je de waarde van *roll* gebruiken die je binnenkrijgt als parameter. Door je mobiel te kantelen ga je de *x*-coördinaat van de raket beïnvloeden met de volgende formule:

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

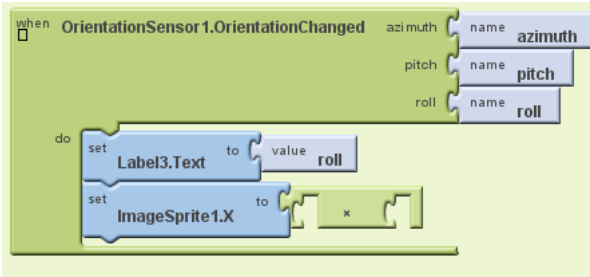
Je hebt dus een *vermenigvuldiging* van een *optelling* van een *deling* nodig en je moet de uitkomst plaatsen in de *x*-coördinaat van de sprite.

### Opgave 5.13

Voer de volgende opdrachten uit.

- Voeg in het `OrientationSensor1.OrientationChanged` block het `Set ImageSprite1.x to` block toe. Deze vind je onder de `My Blocks` tab bij de sprite.
- Voeg in het 'to'-vak allereerst een block toe voor de vermenigvuldiging. Dit vind je onder de `Built-In` tab bij `Math`. In het `×` block zitten twee openingen waar de twee waarden of expressies in moeten die vermenigvuldigd moeten worden.

Je programma zou er nu uit moeten zien zoals afgebeeld in figuur 5.3.



Figuur 5.3 positie

## 5.5 xmax

Je bent nu op het punt aangekomen dat je de variabele *xmax* moet definiëren en gaan bepalen. De definitie van een variabele zet je op een willekeurige plaats in het werkblad. Het block voor variabele definitie vind je bij **Built-in** | **definition** | **def variable as**. Het woord *variable* moet je vervangen door de naam van de variabele; in dit geval dus *xmax*.

### Opgave 5.14

Maak de *variable xmax* aan en geef deze de waarde 0, zie figuur 5.4 voor hoe dit eruit ziet.

Figuur 5.4 global *xmax*

De echte waarde moet *Canvas.Width - ImageSprite.Width* zijn, maar dat mag niet op deze plaats. Tijdens de definitie van een globale variabele mag nog je geen eigenschappen van componenten gebruiken.

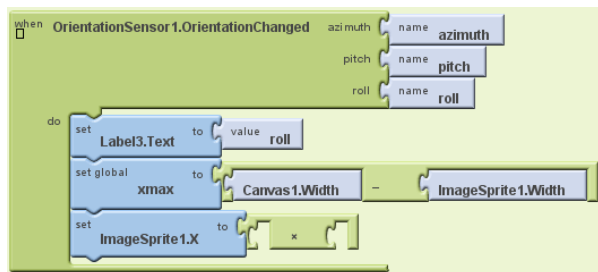
### Opgave 5.15

Probeer het bovenstaande maar eens uit en kijk wat *App Inventor* voor een foutmelding geeft.

Het toekennen van de waarde *Canvas.Width* – *ImageSprite.Width* kun je het beste doen in het **OrientationChanged** block voor je de variabele nodig hebt. Hiervoor gebruik je het **set global xmax to** block dat te vinden is bij **My Blocks** | **My Definitions**. Als het goed is weet je intussen waar je de *width* van het canvas en de sprite kunt vinden en hoe je twee getallen van elkaar aftrekt.

### Opgave 5.16

Geef *xmax* de juist waarde op de juiste plaats. Een voorbeeld van hoe de code eruit kan zien zie je in figuur 5.5.



**Figuur 5.5** berekenen global xmax

Nu je *xmax* de juiste waarde hebt gegeven kun je de variabele gebruiken om de *x*-coördinaat van de raket te bepalen.

**Opgave 5.17**

Maak de code af die de  $x$ -coördinaat uitrekent zoals aangegeven in de paragraaf over de besturing. Het `global xmax` block kun je vinden bij

My Blocks | My Definitions .

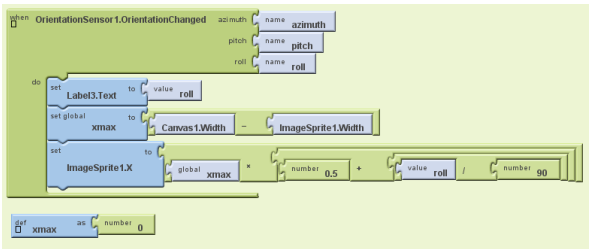
Merk op dat de manier waarop je de blocks in elkaar klikt werkt als het zetten van *haakjes*. Als je de juiste code hebt gemaakt, dan heb je daadwerkelijk het volgende geïmplementeerd:

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

en *niet* bijvoorbeeld:

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

Als het goed is heb je nu code die eruit ziet als in figuur 5.6.



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

Probeer het gebouwde programma uit op je mobiel. Kantel de mobiel en kijk wat de raket doet.



## 5.6 Geen Orientation Sensor?

Als je geen mobiel of Orientation Sensor hebt zul je een andere implementatie moeten vinden om het spel te spelen. In deze paragraaf vind je een alternatieve implementatie die ervoor zorgt dat je het spel kunt spelen op de emulator. Ook als je in de vorige paragraaf de versie met de Orientation Sensor hebt gemaakt, is het interessant om ook deze implementatie te bekijken.

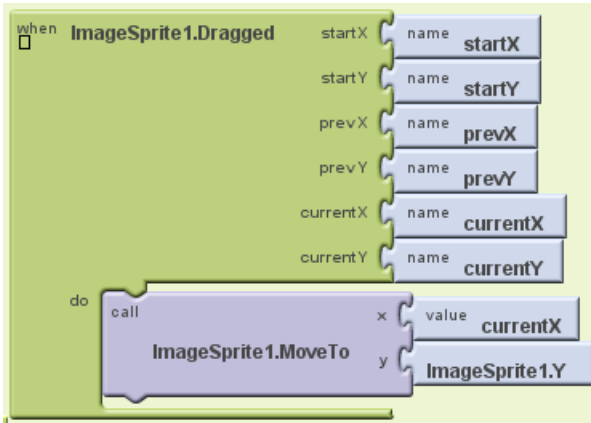
In de **Blocks Editor** bij **My Blocks** | **ImageSprite1** vind je het **ImageSprite1.Dragged** block.

☞ De y-coördinaat moet blijven wat ie was!

### Opgave 5.18

Sleep het **ImageSprite1.Dragged** block in het werkblad en kijk of je snapt wat het doet. Met dit block is het mogelijk de raket te besturen door met je vinger (de muis in de emulator) over het display te 'draggen'. In figuur 5.7 zie je hoe de code eruit zou kunnen zien.





**Figuur 5.7** drag de raket

Als je beide manieren om de raket te besturen tegelijkertijd in je code hebt staan dan geeft dit vreemde effecten als je je mobiel vasthoudt. Het is namelijk best lastig om je mobiel exact horizontaal te houden. Als je hem op tafel legt kan het wel werken. Je kunt een code block uitzetten door het te deactiveren. Dit doe je door met je muis rechts te klikken op de component en dan **Deactivate** te kiezen in het contextmenu. Vooral tijdens het ontwikkelen kan het soms handig zijn iets tijdelijk uit te zetten.

### Opgave 5.19

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

## 5.7 Raket onderaan het scherm

Je hebt nu een raket gemaakt die je kunt besturen, je komt echter nog geen obstakels tegen op jouw weg door het heelal. Daar ga je 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 zodanig dat de raket nog op het canvas staat en zichtbaar is: de waarde van de *y*-coördinaat moet daarom zijn:

$$\text{Canvas.Height} - \text{ImageSprite1.Height}$$

Er is één belangrijke vuistregel bij het programmeren die tot nu toe niet goed is toegepast: de raket heeft namelijk de naam *ImageSprite1*, dit is niet bepaald een duidelijke naam. Zo gauw er meerdere objecten in beeld zijn kan dat natuurlijk niet meer. Je gaat daarom *ImageSprite1* hernoemen naar *Raket*.

### Opgave 5.20

Selecteer *ImageSprite1* (onder *Components*). Gebruik de knop *Rename* 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 hoeft je deze waarde maar één keer te zetten, namelijk bij het initieel opzetten van het scherm. Tijdens het initieel opbouwen (ook wel *initializing* genoemd) van het screen worden de commando's uit het block *Screen1.initialize* (zie *My blocks* | *Screen1*) uitgevoerd.

**Opgave 5.21**

Zet een **Set Raket.Y to**-block in **Screen1.initialize**. Als je dit niet weet te vinden, zoek dan enkele bladzijden terug waar **ImageSprite1.X** vandaan kwam. Bouw vervolgens de volgende formule op:



**Figuur 5.8** Zet de raket onder in het scherm

Test vervolgens of je geslaagd bent.



## 5.8 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 **Ball1** tot **Meteoor**.

**Tip**

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

pagina te bekijken. Naast dat je daar vaak het antwoord op je vraag vindt, is er veel nuttige informatie te vinden.

Als je de metoor wilt bewegen kun je natuurlijk de  $x$  en  $y$ -coördinaten veranderen. Als je echter de metoor selecteert en kijkt in de rechterkolom bij de **Properties** zie je eigenschappen als **Heading**, **Interval** en **Speed**.



### Opgave 5.22

Zet de **Speed** van de metoor op 40 en kijk wat er gebeurt.

Het **Interval** staat waarschijnlijk op 1000. Dat betekent dat elke 1000 milli-seconde (dat is dus 1 seconde) de metoor met zijn snelheid vooruit wordt gezet. Als de beweging schokkerig overkomt kun je het **Interval** kleiner maken.



### Opgave 5.23

Zet de **Interval** op 200 van en kijk wat er gebeurt.

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

### Opgave 5.24

Met behulp van de **Heading** (dit is de 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. De metoor beweegt dan in de richting van de raket.

Als alles tot zover is gelukt zie je dat de meteor erg klein is. 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 en snelheid is.

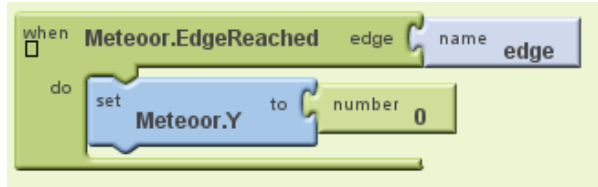
## 5.9 Meerdere meteoren

Als de meteor beneden is aangekomen blijft hij liggen. Je wilt eigenlijk dat de meteor dan weer van boven het scherm binnenkomt, wellicht zelfs in een andere grootte of kleur. Gelukkig helpt *App Inventor* je ook hier weer uit de brand, namelijk met een event `EdgeReached`. Dit event treedt op als de bal de onderkant van het scherm raakt. Je vindt `Meteor.EdgeReached` onder `Meteor` in `My Blocks`.

Als de meteor beneden tegen de rand aan vliegt wil je dat de y-coördinaat weer 0 wordt, zodat de bal weer bovenaan begint. Als je `Meteor.EdgeReached` op het canvas sleept zie je dat deze een parameter `name edge` heeft. Hiermee kun je kijken tegen welke kant de meteor gevlogen is. Aangezien de bal omlaag vliegt hoef je hier niet naar te kijken, je weet immers al dat hij onder is aangekomen.

### Opgave 5.25

Implementeer `Meteor.EdgeReached` zodanig dat de bal weer bovenaan het scherm begint nadat deze de onderkant heeft geraakt.



**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 blijft de meteor steeds op dezelfde plek naar beneden komen. Het schokkerige kun je een eind oplossen door de het **Interval** van de raket kleiner te maken, net als je bij de meteor ook gedaan hebt.



### Opgave 5.26

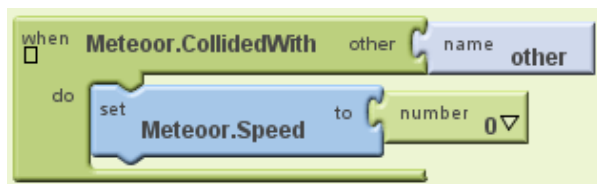
Klik op de raket en zet het interval op 100. Hoe kleiner het getal, hoe vaker de raket getekend wordt en dus hoe soepeler de beweging er uit ziet. Je zult echter wel snappen dat de mobiel het dan ook drukker heeft.

## 5.10 Botsing

Nu ga je kijken naar het botsen. Ook hiervoor bestaat een event, die je vindt onder **My Blocks** | **Meteor** | **Meteor.CollidedWith**.

Het event treedt op als de Meteor tegen een ander voorwerp botst. Uit de parameter *other* kun je bepalen met welk voorwerp de meteor is gebotst. Aangezien je al weet dat het de raket is hoeft je dit niet te controleren. Zodra het

event optreedt weet je dat de raket tegen een meteor is geknald. Voor nu zet je dan de meteor stil, al is dat niet erg spectaculair. Sleep de volgende code bij elkaar:



Figuur 5.10 Botsing

### Opgave 5.27

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. Pas de code aan zodat het ook een geluid laat horen.



Als je de meteor succesvol ontwijkt wil je dat hij op een willekeurige plek bovenin het scherm terugkomt, oftewel in het `EdgeReached` block wil je een willekeurige  $x$ -coördinaat zetten. Iets vergelijkbaars heb je eerder al gedaan!

### Opgave 5.28

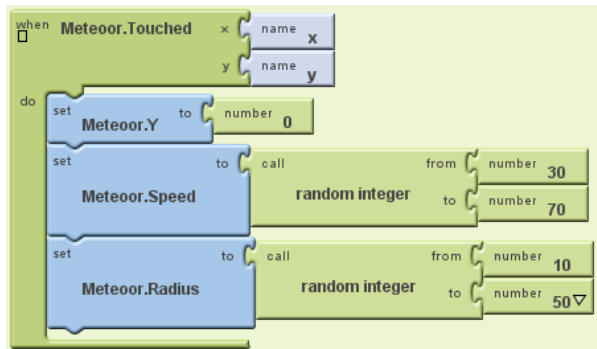
Gebruik `set Meteoor.x to` en `random integer` om de  $x$ -coördinaat een willekeurige waarde tussen 0 en de maximale  $x$ -coördinaat te geven.

☞ Je hoeft de maximale  $x$ -coördinaat niet nogmaals uit te rekenen.

## 5.11 Nog een stap verder



Als de meteor stil staat en je klikt er op (`Meteor.Touched`) moet het spel weer verder gaan. Er moet een *'nieuwe'* meteor van boven komen. Hiertoe moet je de `y` op 0 zetten en de snelheid – die je bij de botsing op 0 hebt gezet – weer op een goede waarde. Je gebruikt een willekeurige waarde tussen 30 en 70 (Waarom?). Verder moeten de meteoren ieder hun eigen grootte krijgen, ook de grootte is dus willekeurig.



Figuur 5.11 Botsing



### Opgave 5.29

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`.



## 5.12 Bonus opgaven

Je hebt inmiddels een leuk spel gemaakt. Uiteraard kun je nog veel aanpassen om het spel leuker te maken. Hieronder vind je enkele ideeën. Voer enkele van deze ideeën uit, maar voel je ook zeker vrij om eigen verbeteringen te implementeren!

### Opgave 5.30

Elke meteor heeft dezelfde kleur, geef iedere meteor een eigen kleur. Bepaal zelf in hoeverre je dit willekeurig maakt.

### Opgave 5.31

Op dit moment komt elke stand van de telefoon overeen met een positie van de raket op het scherm. Pas het spel aan zodat de *roll* de versnelling bepaald: hoe verder de speler de telefoon kantelt, hoe sneller de raket beweegt.

### Opgave 5.32

Net als bij het Mollen Meppen kun je een manier verzinnen om een score bij te houden. Verzin een interessante methode en implementeer deze. Laat uiteraard de score ook live op het scherm zien! Wellicht maak je ook een *high score*?

### Opgave 5.33

Naarmate de speler langer aan het spelen is wil je dat het spel moeilijker wordt. Zorg hiervoor door bijvoor-

beeld na verloop van tijd de snelheid te verhogen of meer meteoren te laten verschijnen.

#### **Opgave 5.34**

Na meerdere keren botsen zou je het spel willen stoppen. Geef de speler drie levens en laat hem/haar bij elke botsing een leven verliezen. Kan hij/zij er ook levens bij verdienen?

# Hoofdstuk 6

## Project

In de afgelopen hoofdstukken heb je drie apps gemaakt in *App Inventor*. Je hebt *App Inventor* nu behoorlijk in de vingers, het is dus hoog tijd om zelf op ontdekkingsreis te gaan! In dit hoofdstuk ga je zelf een app verzinnen en ontwikkelen.

Het is voor dit project aan te raden om *tweetalen* te vormen. Samen weet en kun je immers meer dan alleen!

### 6.1 Brainstorm

Het verzinnen van een app kan best lastig zijn, je begint daarom met een brainstorm sessie. Allereerst bepalen jullie samen het soort app dat je wilt ontwikkelen: een game, een hulpmiddel, een educatieve app? Neem daarna samen met je partner een vel papier en schrijf hier elk idee binnen deze categorie op wat in je opkomt. Houd hierbij de volgende regels in gedachten:

### Brainstorm regels

- Geen enkel idee is te gek
- Slechte ideeën bestaan niet
- Kwantiteit is belangrijk, schrijf dus vooral *veel* op
- Ideeën combineren leidt tot betere ideeën (tip: dit gaat makkelijker als je de ideeën categoriseert)

Nu je een hoop ideeën op papier hebt staan ga je samen een idee uitzoeken om te ontwikkelen. Bedenk hierbij of het haalbaar is (je docent kan je hierin adviseren!) en of het niet te eenvoudig is. Overleg met een ander tweetal over jullie ideeën, wie weet kun je het nog verder aanscherpen!

### Opgave 6.1

Beschrijf jullie idee in een projectvoorstel van een half A4tje. Vermeld hierin tenminste het doel van de applicatie en hoe je dit doel denkt te bereiken. Geef aan welke functies van de telefoon (bijvoorbeeld de orientation sensor) je zult gebruiken. Voeg ook tenminste één schets toe van de gebruikersinterface.

Leg dit voorstel voor aan je docent. De docent zal aangeven of het voorstel haalbaar en voldoende is. Na goedkeuring kun je verder naar de volgende fase.

## 6.2 Ontwerpen

Voordat jullie daadwerkelijk aan de slag gaan ga je nadenken over het ontwerp. Het is een goede gewoonte om voor

je begint met programmeren eerst een ontwerp te maken. Door dit te doen voorkom je dat je tijdens het programmeren tegen onverwachte zaken aanloopt. Ook zal uiteindelijk de structuur van je programma duidelijker worden, wat de code makkelijker te begrijpen maakt voor jullie zelf en voor de docent.

### Opgave 6.2

Werk de schets(en) van de gebruikersinterface verder uit. Zorg ervoor dat je alle toestanden of situaties in de app kunt laten zien.

### Opgave 6.3

Schrijf op welke acties er plaatsvinden bij de verschillende handelingen die een gebruiker kan verrichten. Wees hierbij zo precies mogelijk.

### Opgave 6.4

Bedenk of je, door gebruik te maken van procedures, je ontwerp kunt verbeteren. Zijn er stukjes code die meerdere keren worden uitgevoerd? Heb je te maken met ingewikkelde code die je kunt opdelen? Denk ook na over de naamgeving.

## 6.3 Ontwikkelen

Nu jullie een ontwerp hebben gemaakt is het tijd om de app te ontwikkelen! Dit doe je net als in de voorgaande hoofdstukken *iteratief*. Dat wil zeggen dat je steeds een

klein stukje ontwikkelt (bijvoorbeeld een scherm, of de implementatie van een knop) en vervolgens uittest. Zo blijven de problemen die je tegenkomt zo klein mogelijk.

Jullie gaan programmeren via het zogenaamde *pair-programming*. Eén van de twee zit hierbij achter de knoppen (de *driver*), de ander kijkt over de schouder mee (de *observer*). De *driver* programmeert dus en de *observer* controleert of er geen fouten gemaakt worden. Jullie moeten beiden kunnen programmeren, wissel dus regelmatig van rol!

#### Opgave 6.5

Ontwikkel de app volgens de hierboven beschreven methode.

## 6.4 Testen en verbeteren



Gefeliciteerd, jullie hebben jullie eerst app volledig zelf gemaakt! Het is hoogste tijd om ook anderen met jullie app te laten spelen/werken. Jij weet exact hoe jouw app werkt, de ander echter niet. Door deze testen zul je dus waarschijnlijk bugs en andere verbeterpunten vinden.

#### Opgave 6.6

Laat de app testen door twee andere groepjes (en test natuurlijk ook hun app!) en noteer de verbeterpunten. Verbeter jullie applicatie en herhaal dit proces een paar keer.

## **6.5 Presenteren**

Tot slot presenteren jullie de ontwikkelde applicatie aan de rest van de klas. Bereid hiervoor een korte presentatie en een demo voor.

*Welk tweetal heeft de mooiste app gemaakt?*





[illegible]