Google Assistant

Een voice adventure

Sinds een half jaar heb ik een Google Home in de woonkamer staan. Ondertussen ben ik redelijk bekend met de mogelijkheden, die out-of-the-box geboden worden. Het is dus hoog tijd om eens zelf functionaliteit toe te gaan voegen. Dit heb ik gedaan in de vorm van een 'voice adventure'. Jullie zijn vast wel bekend met het fenomeen text adventure, waarbij je tekst gebruikt om de spelwereld te manipuleren. In het geval van een voice adventure gebruik je dus je stem in plaats van tekst.

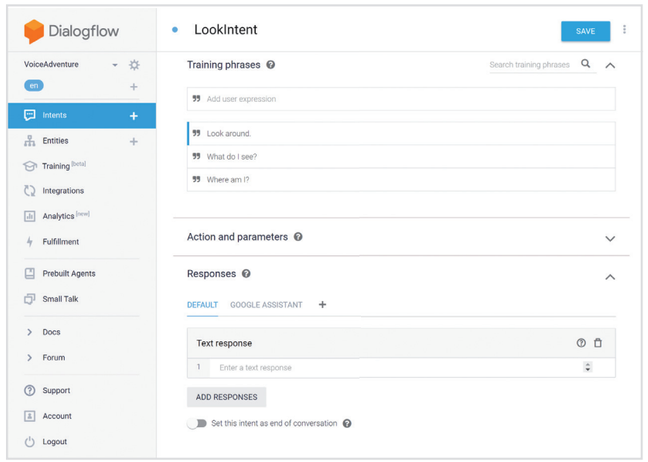
De Google Home is één van de slimme speakers van Google. Het brein van deze speaker is de Google Assistent. Dit is ongeveer dezelfde assistent die je aantreft op moderne Androidtelefoons. Tevens is de Google Assistant beschikbaar voor iPhone in de App Store. Er zijn wat kleine verschillen, maar deze zijn niet relevant voor dit artikel. Het spel werkt dan ook op alle smaken.

Een uitbreiding op de Google Assistent wordt een Assistant app genoemd. Deze kan je aanmaken op een platform genaamd **Actions on Google** (**https://console.actions.google.com**). Hier maak je een project aan en vervolgens dien je daar Actions aan toe te voegen. De makkelijkste manier om dit te doen, is gebruik maken van bestaande templates. Deze stellen je in staat om heel snel en zonder enige programmeerkennis een Action aan te maken. Aangezien dit een artikel is in het Java Magazine, is dit natuurlijk niet de optie die we gaan kiezen. Tevens biedt deze optie ook te weinig vrijheid om het einddoel te bereiken. Oftewel, je dient hier te kiezen voor een custom app. Er zijn weer meerdere mogelijkheden, maar voor het maken van een voice adventure kiezen we voor de Dialogflow optie. Dit is een simpele speech interaction builder, waarin je relatief eenvoudig een app in elkaar kunt klikken. Deze gebruiken we om een eerste opzet van het spel te maken.

Voordat we hier echter mee aan de slag gaan, moeten we eerst even kort schetsen hoe het spel in elkaar zit. Er zijn vijf locaties waartussen we kunnen bewegen. Op deze vijf plekken kan je interacteren met de omgeving door het geven van commando's. Je kunt hierbij denken aan 'look around', 'go north', 'pick up key' en 'use sword'.

Afbeelding 1 geeft een indruk van de Dialogflow interface. Aan de linkerkant zie je de twee belangrijkste concepten waarmee je kunt werken: **intents** en **entities**. Een intent is vergelijkbaar met een request reply paar. Het request is wat de gebruiker zegt en de reply is de reactie, die daarop volgt. Het request kan worden gespecificeerd onder het kopje **Training phrases**. Het reply staat dan onder het kopje **Response**. We negeren nu een aantal opties, maar dit is in essentie een **intent**. Een voorbeeld van een request is, vragen waar

Bouke Nijhuis
is Managing Consultant bij CINQ ICT.



Afbeelding 1: Het LookIntent in Dialogflow

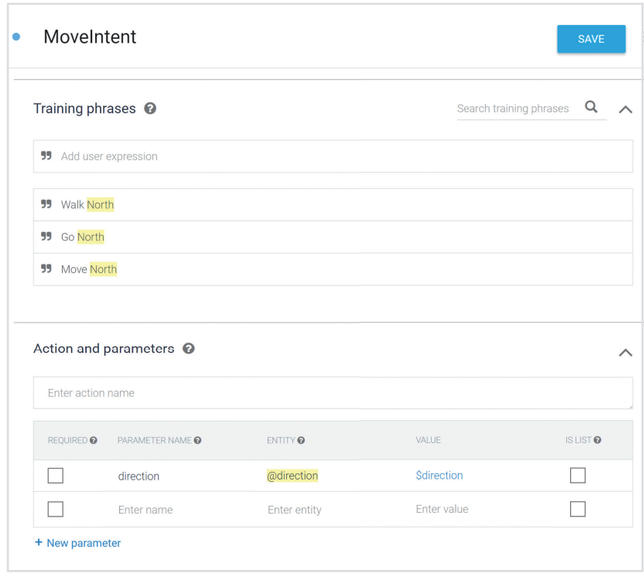
je bent en het antwoord zou vervolgens een beschrijving van de omgeving zijn.

Een entity is een variabele. Deze wordt hoofdzakelijk gebruikt om variabele waarden uit een request te halen. We gaan deze bijvoorbeeld gebruiken om de windrichting te bepalen in het commando 'go north'. In **Afbeelding 2** zie je hoe dit eruit ziet in Dialogflow. In dit geval wordt 'north' toegewezen aan de entity @direction. Je kunt dus ook 'go south' of 'go east' zeggen. Dit geeft een redelijk overzicht wat je allemaal kunt bereiken in Dialogflow. Niet alle concepten zijn behandeld, maar wel de belangrijkste.

Voor het spel moeten we een toestand (state) kunnen bijhouden. We moeten de locatie van de speler bijhouden en kunnen aanpassen. Hetzelfde geldt voor de items in de inventory. Deze gegevens kunnen we opslaan in een entity, maar het manipuleren van deze entities is vrij beperkt in Dialogflow. De gemakkelijkste manier om de locatie op te slaan, is door middel van een x- en y-coördinaat. Als de speler vervolgens beweegt, dan zullen we een coördinaat moeten aanpassen. Echter, dit soort simpele berekeningen kun je niet uitvoeren binnen deze speech interaction builder.

Het probleem dat Dialogflow niet krachtig genoeg is, komen we op veel meer plekken tegen. Je kunt hierbij denken aan onder andere het bepalen van de grenzen van de speelwereld en het bijhouden van de inventory. Eigenlijk in alle gevallen waarin je de toestand moet bijhouden, is custom Java-code nodig. Gelukkig is het niet moeilijk om eigen code toe te voegen aan een Assistant app.

Het is nu tijd om de scheidslijn tussen Dialogflow en custom code te bepalen. De eerste is erg goed in het vertalen van spraak naar tekst en het herkennen van een request. Hiervoor gaan we dus de speech interaction builder gebruiken. Voor alle andere zaken hebben we custom code nodig. Het toevoegen van custom code gaat door middel van een webservice. Eerder is uitgelegd dat een intent vergelijkbaar is met een request reply paar. In Dialogflow kun je een statisch reply definiëren. Je kunt het antwoord echter ook laten geven door een webservice. Dit wordt een fullfilment genoemd. Per intent kun je aangeven of je gebruik wilt maken van het statische antwoord of van de fullfilment. Bij de fullfilment kan je één URL definiëren waar de requests worden heen gestuurd aan



Afbeelding 2 - het MoveIntent in Dialogflow

```
public void handleRequest(InputStream inputStream, OutputStream
outputStream, Context context) throws IOException
{
   // handle request
}
```

Listing 1

de hand van een HTTP POST. Dit endpoint is vervolgens verantwoordelijk voor het geven van een antwoord.

Voor het afhandelen van deze requests zijn serverless function frameworks uiterst geschikt. Hierbij kun je denken aan AWS Lambdas, Google CloudFunctions & Azure Functions. Voor de implementatie van dit spel is er gebruik gemaakt van een AWS Lambda. Je kunt hier een simpele .jar uploaden en vervolgens aangeven welke methode er moet worden aangeroepen. Deze "main-methode" staat in **Listing 1**.

Het InputStream object bevat het request dat is verstuurd vanuit Dialogflow. Het antwoord wordt uiteindelijk gegeven via de OutputStream. Het Context object kan worden gebruikt om AWS-zaken aan te spreken. Hierbij kun je denken aan het uitvragen van informatie over de Lambda of het verkrijgen van een AWS Logger object. De requests komen binnen in de vorm van een JSON met daarin allerlei (context) informatie over het request.

De belangrijkste velden hierin zijn:

- **resolvedQuery**: het exacte commando van de gebruiker.
- **intentName**: de naam van het herkende intent (bijvoorbeeld een LookIntent, zoals in **Afbeelding 1**).
- **parameters**: ingevulde entities (bijvoorbeeld 'north' als windrichting in **Afbeelding 2**).
- **contexts**: opgeslagen gegevens over voorgaande interacties.

In de praktijk maak je geen gebruik van de resolvedQuery, maar handel je alles af op basis van de herkende intent in combinatie met de parameters en de contexts. Voor het maken van een Voice Adventure heb je ongeveer tien verschillende intents nodig. De meest sprekende voorbeelden zijn 'bewegen' en 'gebruiken'. Deze twee intents zullen verderop in detail worden uitgewerkt, maar voor dit mogelijk is, moet er eerst dieper worden ingegaan op de spelwereld.

De spelwereld bestaat uit locaties met daarin voorwerpen. Elke locatie heeft een positie en een omschrijving. De positie is nodig voor het bepalen van aan elkaar grenzende locaties. De omschrijving wordt gebruikt om de omgeving te beschrijven. De voorwerpen zijn iets complexer. Deze hebben naast een locatie en beschrijving ook een indicator of ze op te pakken zijn. Tot slot is er nog een datastructuur, die aangeeft welke voorwerpen op elkaar gebruikt kunnen worden en wat de effecten daarvan zijn.

De eerste intent die we in detail gaan bespreken, is de **MoveIntent**. In **Afbeelding 2** zie je met welke commando's deze intentie wordt getriggerd. Wanneer je 'move' of 'go' combineert met een windrichting, dan herkent Dialogflow dit als een MoveIntent. Alles wat hierop lijkt, zal als een MoveIntent worden geïnterpreteerd. Zo zal bijvoorbeeld 'run south' ook als een MoveIntent worden opgepakt. Als gevolg hiervan wordt er een request naar het ingestelde endpoint gestuurd. Een voorbeeld van een versimpeld request zie je in **Listing 2**.

Het eerste wat de software doet, is achterhalen om welk intent het gaat. In dit geval is het een MoveIntent en wordt de zogenoemde MoveHandler class gebruikt om het request af te handelen. Voor elke intent is er een bijpassende handler class. Deze is dan verantwoordelijk voor het bijwerken van de toestand (state) en het formuleren van een antwoord.

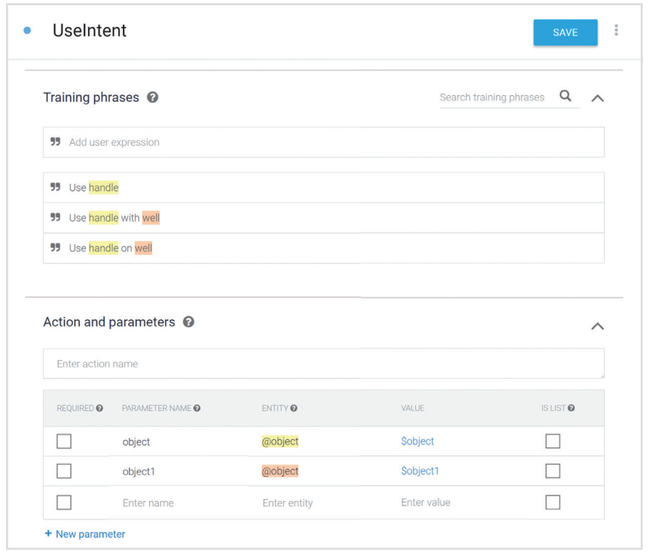
In het geval van het bovenstaande request zal de MoveHandler allereerst kijken of de verplaatsing mogelijk is. Hiervoor wordt eerst gekeken naar de huidige locatie (van de speler) en de gekozen richting. Als de verplaatsing mogelijk is, dan wordt de huidige locatie aangepast en krijgt de speler een beschrijving van de nieuwe locatie. Als dit niet mogelijk is,

```
{
 "result": {
   "resolvedQuery": "move north",
   "parameters": {
     "direction": "N"
   },
   "contexts": [],
   "metadata": {
     "intentName": "MoveIntent"
   }
 }
}
```

Listing 2

```
{
 "speech": "You are standing in front of a big castle.",
 "displayText": "You are standing in front of a big castle.",
 "contextOut": [
   {
     "name": "state",
     "parameters": {
       "posx": 0,
       "posy": 1
     }
   }
 ]
}
```

Listing 3



Afbeelding 3: het UseIntent in Dialogflow

dan krijgt hij hier een melding van (en wordt de huidige locatie natuurlijk niet aangepast). Het bijbehorende (versimpelde) reply staat in **Listing 3**.

De waarde van de attributen **speech** en **displayText** zijn gelijk. Dit houdt in dat je het spel kunt spelen met voice- als text-interactie. De toestand wordt bijgehouden in een context(Out) object, genaamd state. Hierin zie je de locatie van de speler als een coördinaat. Deze was in het request niet gedefinieerd en wordt default vervolgens op (0, 0)gezet. Door een succesvolle beweging naar het noorden is het (0, 1) geworden.

Een ander interessant intent is de **UseIntent**. Deze wordt gebruikt wanneer er voorwerpen op elkaar worden gebruikt. Een voorbeeld hiervan is het zwaard op de trol gebruiken door middel van'use sword on troll'. In dit geval zijn er dus twee parameters (namelijk sword en troll). In **Afbeelding 3** zie je hoe dit werkt in Dialogflow. Er zijn dan twee parameters van de entiteit @object gemaakt.

Deze twee (ingevulde) parameters zie je vervolgens weer terug in het versimpelde request in **Listing 4**. Wanneer je dit request vergelijkt met het vorige request in **Listing 2,** dan zie je dat nu het contexts-veld ook is gevuld. De inhoud lijkt erg op de inhoud van het veld contextOut in **Listing 3**. Het is wederom het context object waarin de state wordt bijgehouden. Er zijn twee nieuwe attributen bij gekomen, namelijk inventory en removedItems. De eerste houdt bij welke objecten de speler heeft opgepakt en de tweede bevat de objecten, die de speler heeft gebruikt in het spel, maar nu niet meer bruikbaar zijn.

Wanneer er een UseIntent binnenkomt, dan wordt er eerst gekeken of beide objecten gevuld zijn. Als dit niet het geval is dan krijgt de speler een melding dat de actie niet kan worden uitgevoerd. Vervolgens wordt er gekeken of beide items op elkaar gebruikt kunnen worden. Hiervoor worden een aantal controles uitgevoerd. Daarbij kun je denken aan zaken als: heeft de speler het object ooit opgepakt, staat de speler op de juiste locatie en heeft het zin om beide objecten op elkaar te gebruiken. Als alle controles positief zijn, dan wordt de actie uitgevoerd en heeft de speler een nieuw object verkregen. Er wordt dan een nieuw object toegevoegd aan het veld inventory en het gebruikte object verhuist van inventory naar removedItems. In alle andere gevallen krijgt

```
{
 "result": {
   "resolvedQuery": "use sword on troll",
   "parameters": {
     "object": "sword",
     "object1": "troll"
   },
   "contexts": [
     {
       "name": "state",
       "parameters": {
         "posx": -1,
         "posy": 0,
         "inventory": ["SWORD"],
         "removedItems": ["HANDLE"]
       }
     }
   ],
   "metadata": {
     "intentName": "UseIntent"
   }
 }
}
```

Listing 4

de gebruiker een melding met daarin de reden waarom de actie niet mogelijk is.

Tot zover de uitleg over het maken van een Assistant app door middel van Dialogflow en custom Java-code. Tot slot wil ik jullie graag uitnodigen om het spel te spelen. Dit doe je door de QR-code in **Afbeelding 4** te scannen. Het kan gespeeld worden met zowel tekst- als spraakinvoer. Uiteraard is de laatste optie de leukste manier en dit doe je door rechts onderin op de microfoon te klikken. Mocht je geen idee hebben hoe te beginnen, dan kun je simpelweg om hulp vragen in het spel. Veel plezier met spelen!

Mocht je na het spelen geïnteresseerd zijn in de bijbehorende Java-code, dan staat er in de referenties een URL naar het GitHub project. ■

Afbeelding 4: QR code

REFERENTIES

Actions on Google: **https://console.actions.google.com**
Dialogflow: **https://console.dialogflow.com/api-client/**
WebDemo: **https://bot.dialogflow.com/4df6f607-bedc-4393-bd59-7485757110ba**
Github: **https://github.com/BoukeNijhuis/google-home-demo**