Universiteit Leiden
The Netherlands

Opleiding Informatica

Scratch of Scottie Go!: een vergelijking tussen twee manieren om te leren programmeren

Okke Moison

Supervisors:
Marileen Smit & Felienne Hermans

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

21/01/2021

Samenvatting

Scratch en Scottie Go! zijn twee manieren waarmee kinderen kunnen leren programmeren. Scratch, een digitale programmeeromgeving, staat bekend als dé programmeertaal voor kinderen. Daartegenover staat Scottie Go!, wat een relatief nieuw spel is waarmee programmeren geleerd kan worden. Dit spel is deels digitaal en deels fysiek. In dit onderzoek vragen we ons het volgende af: is Scratch of Scottie Go! beter om kinderen te leren programmeren? De voor- en nadelen van de beide talen (voor de leesbaarheid zal Scottie Go!, net als Scratch, vaak een taal genoemd worden) worden in dit onderzoek besproken aan de hand van het cognitive dimensions of notations framework en algemene eigenschappen. Uit deze theoretische analyse blijkt dat Scottie Go! meer geschikt is voor complete beginners, terwijl Scratch meer voordelen heeft voor gebruikers die de basis kennen. Op basis van deze analyse kan geconcludeerd worden dat de twee juist op elkaar aansluiten in plaats van dat de talen concurreren. Naast de theoretische analyse is er een vooronderzoek voor een experiment gedaan, waarin onderzocht is in hoeverre het mogelijk is om Scratch en Scottie Go! in de praktijk te vergelijken. Dit is gedaan door de levels van Scottie Go! om te zetten naar Scratch. Hoewel we bij de vertaling tegen de nodige problemen opliepen, hebben we de nodige inzichten gekregen en ideeën opgedaan om in een dergelijke richting verder te onderzoeken.

Inhoudsopgave

1 Inleiding **1**

2 Gerelateerd Werk **1**

3 Scottie Go! **2**

4 Scratch **3**

5 Cognitieve Dimensies **4**

5.1 Closeness of Mapping . . . 5
5.2 Viscosity . . . 5
5.3 Hidden Dependencies . . . 6
5.4 Hard Mental Operations . . . 6
5.5 Premature Commitment . . . 7
5.6 Secondary Notation . . . 8
5.7 Visibility . . . 8
5.8 Error-Proneness . . . 9
5.9 Abstraction . . . 9
5.10 Role-Expressiveness . . . 10
5.11 Consistency . . . 10
5.12 Diffuseness . . . 10
5.13 Provisionality . . . 11
5.14 Progressive Evaluation . . . 11

6 Scratch of Scottie Go! om te Leren Programmeren **12**
6.1 Closeness of Mapping . . . 12
6.2 Viscosity . . . 12
6.3 Hidden Dependencies . . . 13
6.4 Hard Mental Operations . . . 13
6.5 Premature Commitment . . . 13
6.6 Secondary Notation . . . 13
6.7 Visibility . . . 14
6.8 Error-Proneness . . . 14
6.9 Abstraction . . . 14
6.10 Role-Expressiveness, Consistency en Diffuseness . . . 14
6.11 Provisionality . . . 14
6.12 Progressive Evaluation . . . 15
6.13 Conclusie cognitieve dimensies . . . 15
6.14 Instructie . . . 15
6.15 Feedback . . . 16
6.16 Conclusie . . . 16

7 Vooronderzoek **16**
7.1 Opzet . . . 17
7.2 Module 1 . . . 17
7.3 Module 2 . . . 18
7.4 Module 3 . . . 20
7.5 Module 4 . . . 20
7.6 Algemene Verschillen . . . 21
7.7 Cognitieve Dimensies . . . 23
7.8 Tests . . . 23

8 Conclusie en Verder Onderzoek **25**

Appendices **28**

A Scratch **28**
A.1 Beweging . . . 28
A.2 Uiterlijken . . . 28
A.3 Geluid . . . 29
A.4 Gebeurtenissen . . . 29
A.5 Besturen . . . 29
A.6 Waarnemen . . . 30
A.7 Functies . . . 30
A.8 Variabelen . . . 31

B Scottie Go! **32**
B.1 Begin and End . . . 32
B.2 Action Tiles . . . 32
B.3 Loops and Conditional Statements . . . 32

B.4 Logic 33
B.5 Switching Characters 33
B.6 Functions 33
B.7 Objects on the Screen 34
B.8 Digits 34

1 Inleiding

Kinderen beginnen steeds vaker op jonge leeftijd met programmeren. Sommige van deze kinderen tonen er zelf interesse in of komen ermee in aanraking via hun ouders. Anderen krijgen misschien een cursus of lessen in programmeren op de basisschool, waar dat steeds vaker aangeboden wordt. In het curriculum van het basisonderwijs en de onderbouw van het voortgezet onderwijs dat op dit moment wordt herzien is programmeren zelfs opgenomen in het conceptprogramma [Cur]. Het is daarom belangrijk om te onderzoeken hoe en waarmee kinderen het beste kunnen leren programmeren. Zeker vergeleken met klassieke vakgebieden zoals rekenen, taal en spelling is daar namelijk weinig over bekend. In deze scriptie zal worden onderzocht wat de voor- en nadelen zijn van twee manieren om te leren programmeren: is Scratch of Scottie Go! beter om kinderen te leren programmeren? Scratch staat al langer bekend als dé programmeertaal voor kinderen. Deze visuele taal is gebaseerd op het gebruik van "blokken" waarmee een programmeur figuren, genaamd "sprites", kan besturen. Gebruikers kunnen zelf aan de slag gaan of ze kunnen gebruik maken van de tutorials die in Scratch ingebouwd zijn. Scottie Go! is vergeleken met Scratch nieuwer. Het is een spel dat is ontworpen om de speler kennis te laten maken met programmeren en hen de basis daarvan te leren. De speler heeft een app op zijn telefoon of tablet en een fysiek speelbord waar fysieke kaartjes met code opgelegd kunnen worden. In Sectie 3 en 4 gaan we dieper in op Scottie Go! en Scratch.

Om antwoord te kunnen geven op de onderzoeksvraag zijn Scratch en Scottie Go! met elkaar vergeleken. Dit is gedaan aan de hand van het cognitieve dimensies framework, waarmee het design van de twee oude de loep wordt genomen. Deze theoretische vergelijking wordt in Sectie 5 beschreven. De gevolgen die deze verschillen hebben op het leerproces worden beschreven in Sectie 6. Bij de vergelijking in die sectie zullen daarnaast ook enkele algemene kenmerken van beide talen besproken worden. Om Scratch en Scottie Go! ook in de praktijk met elkaar te kunnen vergelijken is een aantal modules van Scottie Go! vertaald naar Scratch. Op die manier zijn er minder variabelen, waar rekening mee moet worden gehouden, als een vergelijking wordt gemaakt tussen kinderen die leren programmeren met respectievelijk Scratch en Scottie Go!. In Sectie 7 wordt hier dieper op ingegaan.

2 Gerelateerd Werk

Scratch en Scottie Go! gaan uit van een een verschillende manier van leren. Scottie Go! is min of meer gebaseerd op het principe van directe instructie. Daar krijgt de speler stap voor stap uitleg over de concepten die hij moet leren en wordt ermee geoefend. Scratch is gebaseerd op het principe ontdekkend leren. Daarbij wordt er geen of weinig uitleg gegeven. Hierbij leert hij door verschillende dingen uit te proberen en te onderzoeken wat wel en niet werkt. Verschillende onderzoeken laten zien dat directe instructie effectiever is dan ontdekkend leren [SJJ04] [KSC06].

Volgens Meerbaum-Salant, Armoni en Ben-Ari [MAB13] is het met Scratch goed mogelijk om kinderen informaticaconcepten aan te leren. Dat zou dan wel moeten gebeuren aan de hand van lesmateriaal dat het leerproces leidt. Wanneer kinderen zonder sturing ermee aan de slag gaan leren ze er niet veel van. In een onderzoek van Videnovik en Trajkovik [VT18], waar Scottie Go! gebruikt werd voor een vak computational thinking, kwamen goede resultaten naar voren. Hier beantwoordde ten minste 90% van de kinderen de vragen correct voor iedere geteste categorie.

3 Scottie Go!

Scottie Go! is een educatief spel ontworpen voor kinderen vanaf zes jaar. Het spel laat kinderen spelenderwijs kennismaken met programmeren. Het is recent ontwikkeld door de Poolse startup BeCreo Technologies die innovatieve onderwijsmiddelen ontwikkelt in samenwerking met leraren, experts, onderwijsinstellingen en research & development entiteiten. Het spel is beschikbaar in 23 talen en in meer dan 70 landen [Goa]. Voor kinderen, die tegenwoordig veel online zijn en games spelen, is het vaak een positieve toevoeging aan het leerproces om te leren aan de hand van een game [VT18]. Naast de lesstof die behandeld wordt hebben games ook nog het voordeel dat het tegelijkertijd ook andere vaardigheden kan aanleren, zoals probleem oplossend denken, besluitvorming en strategisch plannen. Daarnaast houden kinderen positieve gedachten over aan programmeren doordat het aangeleerd wordt aan de hand van een game. In een coöperatief spel zoals Scottie Go! helpen leerlingen elkaar met de opdrachten die zij moeten oplossen, zoals ook in een flipped classroom het geval is. Daar ligt de focus van het leren op de leerlingen en hoe zij samen tot een oplossing van de opdrachten komen, de leraar is dan vooral bezig met het proces van de leerlingen in goede banen te leiden en te helpen waar dat nodig is [VT18]. In het design van Scottie Go! zijn deze dingen terug te vinden: het leert kinderen aan de hand van een game programmeren en raadt aan om dit in groepjes te doen.

Scottie Go! bestaat uit een app, een speelbord en kartonnen tegels. De tegels en het speelbord dienen samen aangeschaft te worden; de app is wel gratis te downloaden, maar dat alleen is niet genoeg om het spel te kunnen spelen. De tegels zijn terug te vinden in Appendix B. Op de app krijgt de speler levels te zien, waarbij de speler een opdracht krijgt om het hoofdpersonage van het spel, Scottie, bepaalde handelingen uit te laten voeren (bijvoorbeeld over alle kruisjes lopen, zoals in Figuur 1). Deze handelingen moet de speler dan programmeren door kartonnen tegels in de goede volgorde op het speelbord te leggen en deze oplossing in te scannen met de app [Gob]. Wanneer de code programmeertechnisch correct is, ziet de speler in de app hoe Scottie zijn code uitvoert. Als de code een goede oplossing is, krijgt de speler één tot drie sterren. Voor een optimale oplossing worden drie sterren gegeven, voor een goede, maar minder efficiënte oplossing twee sterren en slechts één ster voor een oplossing die te lang was [Zom19a]. Als de code fouten bevat, krijgt de speler een foutmelding te zien waar ook beschreven is waar de fout zit. Als de code wel programmeertechnisch correct is, maar niet een oplossing van het level krijgt de speler ook een foutmelding te zien, dit keer met een tip. De speler ondervindt geen nadelen aan het maken van fouten. Het level kan altijd opnieuw geprobeerd worden, ook als het al is gehaald. In Figuur 1 is een voorbeeld van een level te zien, inclusief een mogelijke oplossing.

Scottie Go! is onderverdeeld in tien modules die ieder bestaan uit acht tot tien levels. In iedere module worden een of meerdere nieuwe programmeerconcepten of onderdelen van het spel uitgelegd, zoals herhaal-lussen, if-else-statements, voorwerpen oppakken en springen over valstrikken. De toepassing hiervan wordt vervolgens in de levels erna moeilijker gemaakt en gecombineerd met eerdere concepten [Zom19b].

Figuur 1: Level 10 van module 1 met een mogelijke oplossing

4 Scratch

Scratch is een programma dat gebruikt kan worden om interactieve verhalen, spellen en animaties te maken. Het is ontwikkeld voor kinderen tussen de acht en zestien jaar door de Lifelong Kindergarten Group van het MIT Media Lab en is sinds mei 2007 te gebruiken. De bedenkers wilden het makkelijker maken voor iedereen om te starten met programmeren en hun creaties te delen met elkaar, zelfs voor mensen die zichzelf niet eerder als programmeur zagen [Res+09]. Ondertussen is Scratch beschikbaar in meer dan 150 landen en in meer dan 40 verschillende talen [Scr]. De naam "Scratch" is afgeleid van het "scratchen" van platen door hiphop diskjockeys, die zo met muziek knutselen om nieuwe muziek te maken. In Scratch wordt volgens de bedenkers op een vergelijkbare manier geknutseld met afbeeldingen, animaties, foto's, muziek en geluiden. Scratch is online[1] te gebruiken, maar het is ook mogelijk om het te downloaden zodat het zonder internetconnectie gebruikt kan worden. Beide opties zijn gratis.

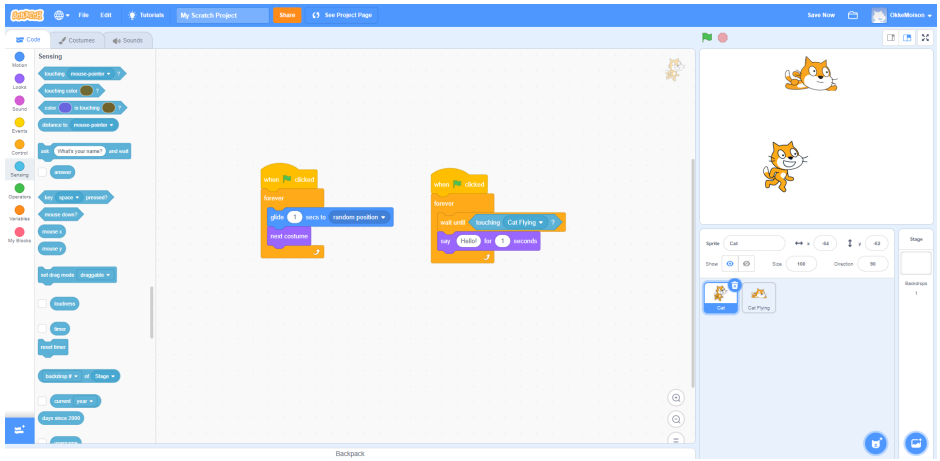
Programmeren in Scratch wordt gedaan door verschillende blokken aan elkaar vast te maken, in tegenstelling tot de meeste programmeertalen is het dus niet mogelijk om code te typen. De blokken zijn terug te vinden in Appendix A, waar ze onderverdeeld staan in de categorieën die Scratch zelf gebruikt. Voor de vormgeving van de blokken hebben ze zich laten inspireren door LEGO: de blokken passen net als LEGO-stukjes op elkaar. De verschillende vormen van blokken geven de gebruiker een hint over of het zal passen of niet. Net als met LEGO is het mogelijk om zomaar te beginnen: zonder voorkennis kan een gebruiker iets proberen met Scratch en zien wat er gebeurt [Res+09]. Dit wordt ontdekkend leren genoemd; leren door te doen, plezier te hebben en te leren van fouten. Deze manier van leren is gebaseerd op de ideeën en filosofie van Seymour Papert [Pap99].

Programmeren in Scratch werkt door code te schrijven voor verschillende 2D figuren, genaamd "sprites". Er zijn veel mogelijkheden om deze sprites te programmeren, ze kunnen bijvoorbeeld bewegen, verschijnen en verdwijnen, spreken met spraakwolkjes en geluiden maken. Daarnaast kunnen ze geprogrammeerd worden om te reageren op de input van een gebruiker die het programma uitvoert om zo een interactieve animatie of een spelletje te maken.

In Figuur 2 is de programmeeromgeving van Scratch weergegeven met een klein voorbeeldprogramma.

[1] https://scratch.mit.edu/

Helemaal links zijn de verschillende categoriën blokken te zien, met direct daarnaast een deel van de blokken die onder de categorie “Waarnemen” vallen. Het grote veld in het midden van het scherm is het scriptveld, daar kan de daadwerkelijke code geschreven worden. Rechtsonder is een overzicht te zien van de sprites en achtergronden die in dit project gebruikt worden. Iedere sprite en achtergrond heeft zijn eigen code, deze kunnen dus vergeleken worden met losse files. Om een nieuwe sprite toe te voegen kan een sprite gekozen worden die standaard in Scratch aanwezig is of er kan een eigen afbeelding gekozen worden. Als laatste is rechtsbovenin het resultaat van de code te zien: wanneer het programma of een stuk code uitgevoerd wordt, toont Scratch dat direct in dat scherm. Een van de features van Scratch is dat de code altijd kan blijven lopen, zelfs als er ondertussen dingen aangepast of toegevoegd worden [Mal+10].



Figuur 2: De programmeeromgeving van Scratch

Daarnaast is voor Scratch ook het sociale aspect van belang: programma’s die in Scratch zijn geschreven kunnen gedeeld worden met andere “Scratchers”. Gedeelde projecten zijn te vinden op de website, waar ze kunnen worden uitgevoerd door iedereen. Daarnaast is het ook mogelijk om de code zelf te bekijken of zelfs te kopiëren om aan te passen [Res+09].

5 Cognitieve Dimensies

Programmeertalen kunnen op verschillende manieren vergeleken worden. Eén van die manieren is het cognitieve dimensies framework. Dit framework is ontwikkeld om discussies over het design van programmeertalen en -omgevingen makkelijker en gestructureerder te maken [BG03]. De cognitieve dimensies kunnen alleen gebruikt worden om verschillende talen te vergelijken, niet om bijvoorbeeld een kwantitatieve waarde toe te kennen aan een kenmerk van een taal [GP96]. In dit geval gaat het dus alleen over hoe Scratch en Scottie Go! ten opzichte van elkaar staan per dimensie. De verschillende dimensies waar het framework uit bestaat beschrijven ieder een kenmerk van de opbouw van een programmeertaal of -omgeving. Al deze kenmerken staan min of meer los van elkaar, maar ze zijn niet geheel onafhankelijk: wanneer een dimensie van een taal veranderd wordt zal ten minste één andere dimensie ook veranderen. Het is dus wel mogelijk dat wanneer een dimensie veranderd wordt een speciefieke andere dimensie niet verandert, zo lang een derde dimensie dat wel doet. Vergelijk dit bijvoorbeeld met de lengte, breedte en oppervlakte van een rechthoek: als de

lengte verandert, kan gekozen worden om de breedte of de oppervlakte gelijk te houden, maar niet allebei [GP96].

In Sectie 7 wordt een praktisch vooronderzoek beschreven waar een aantal modules van Scottie Go! nagemaakt is in Scratch. Dit heeft gevolgen voor de cognitieve dimensies in Scratch. Deze gevolgen zullen worden besproken in Sectie 7.7. Hier in Sectie 5 worden de cognitieve dimensies besproken zoals deze gelden voor Scratch en Scottie Go in z'n geheel. In Tabel 1, aan het eind van deze Sectie, staat per dimensie aangegeven welke programmeertaal hoger scoort. Houd in gedachten dat de taal die vergeleken met de ander hoger scoort in een bepaalde dimensie niet per se beter of slechter is. In Tabel 1 zijn de vergelijkingen samengevat. Hou in gedachten dat de taal die vergeleken met de ander hoger is in een bepaalde dimensie niet per se beter of slechter is.

5.1 Closeness of Mapping

Closeness of mapping is de diminsie die aangeeft hoe dicht een programmeertaal ligt bij het domein waar de taal voor gebruikt wordt. [GP96]. Neem bijvoorbeeld C en Scratch. C is een taal die ontwikkeld is zonder een specifieke toepassing, de syntax is zeer algemeen en kan gebruikt worden voor veel verschillende taken. Daardoor is er vrijwel geen direct verband tussen de syntax en de programma's die in C geschreven worden. Scratch heeft daarentegen een minder algemene syntax. In Scratch is er een aantal instructies, vooral de blokken die onder "Beweging" en "Besturen" vallen (Appendix A.1 en A.5), die direct in verband staan met de sprites die Scratch gebruikt. Hieruit blijkt dat de closeness of mapping voor Scratch groter is dan dat van C.

De closeness of mapping voor Scottie Go! is hoger dan die voor Scratch. Dit volgt uit het feit dat Scottie Go! gemaakt is om levels in het spel op te lossen, alle mogelijke instructies zijn daarom direct gerelateerd aan het spel. Scratch is niet gemaakt om levels in een spel op te lossen, maar juist om programma's zoals spellen en animaties zelf te maken. De closeness of mapping is daardoor minder, omdat het domein per programma verschilt.

5.2 Viscosity

Viscosity gaat over de moeite die de gebruiker van een taal moet doen voor een kleine aanpassing van de code. Die moeite kan per aanpassing verschillen [GP96].

Scratch heeft voor loops en conditional statements verschillende soorten blokken. Zo onderscheidt Scratch drie soorten herhalingen, zoals te zien in Figuur 3.

Figuur 3: Drie soorten herhalingen: oneindig, X keer en conditioneel

Wanneer de gebruiker een herhaalblok wilt omwisselen voor een ander herhaalblok moet hij de blokken erboven, eronder en erin eerst loshalen, vervolgens het oude herhaalblok verwijderen, het nieuwe herhaalblok pakken en daarna nog alle losgekoppelde blokken terugzetten om het nieuwe

herhaalblok heen. In het geval van conditionele blokken geldt bijna hetzelfde. Daar zijn in Scratch twee variaties van, namelijk “als ... dan” en “als ... dan - anders”. Ook hier moet het volledige blok worden omgewisseld als de gebruiker van de ene naar de andere wilt wisselen.

In Scottie Go! is het vervangen van een herhaalblok een stuk makkelijker. Daar hoeft de speler alleen maar het ene herhaalblok om te wisselen voor het andere herhaalblok samen met de condities die bij de blokken horen. De blokken die boven of onder het herhaalblok stonden hoeven dus niet verschoven te worden. Voor conditionele statements geldt ook dat het op dezelfde manier te vervangen is, bijvoorbeeld een “ELSE”-blok door een “ELSE-IF” blok. Verder kan een dergelijk blok ook makkelijk onderaan al neergelegde blokken toegevoegd worden.

Voor andere aanpassingen is het verschil niet altijd zo groot als bij conditionele stataments of herhalingen. In het algemeen wanneer een blok wordt toegevoegd tussen andere blokken worden die blokken in Scratch automatisch verschoven. In Scottie Go! moeten de blokken wel handmatig verschoven worden, dus dat is iets meer werk. Aan de andere kant kan een blok in Scottie Go! gemakkelijk tussen andere blokken uitgehaald worden, waarna de overige blokken tegen elkaar aan geschoven kunnen worden. In Scratch moeten de blokken boven en onder het te verwijderen blok eerst losgehaald worden en pas daarna kunnen de overige blokken weer aan elkaar vastgemaakt worden. Alles bij elkaar genomen is het dus makkelijker om code te veranderen in Scottie Go!, het heeft een hogere viscosity dan Scratch.

5.3 Hidden Dependencies

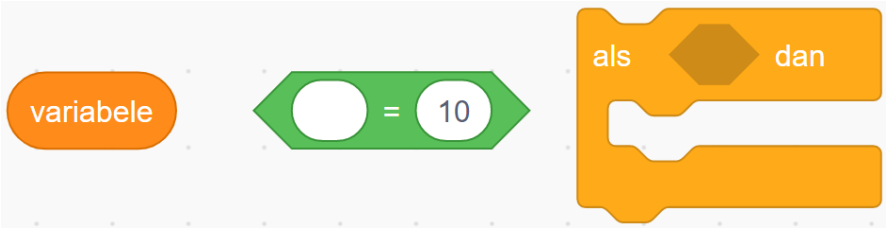
Hidden dependencies zijn verbanden in een programma die niet altijd duidelijk zichtbaar zijn [GP96]. Voor Scratch is het zo dat het niet altijd duidelijk is door welke sprites bepaalde variabelen veranderd worden. Globale variabelen in Scratch kunnen namelijk door iedere sprite aangepast worden, terwijl het niet mogelijk is om in Scratch alle sprites te doorzoeken naar een bepaalde variabele. Wanneer bijvoorbeeld sprite A gebruik maakt van een variabele uit sprite B, terwijl die weer gebruikt maakt uit een variabale uit sprite C, kan een verandering van die laatste variabele gevolgen hebben voor de werking van sprite A, terwijl het niet direct duidelijk is dat dat de oorzaak is. Zeker in grotere programma’s zijn hidden dependencies terug te vinden.

Scottie Go! heeft helemaal geen hidden dependencies. Alle code voor de oplossing van een bepaald level wordt op hetzelfde bord neergelegd, dus er is helemaal niks verborgen. Scratch heeft dus meer hidden dependencies.

5.4 Hard Mental Operations

Hard mental operations, ofwel moeilijke mentale activiteiten, heeft betrekking op combinaties van primitieven. Primitieven zijn de kleinste onderdelen van een taal [GP96]. In het geval van Scratch en Scottie Go! gaat het dus om de mate van mentale moeite die het kost om blokken te combineren. Zowel het design van Scratch als dat van Scottie Go! is gericht op kinderen, bij beide is dus veel aandacht besteed aan het design van de blokken om het schrijven van de code logischer te maken. Beide gebruiken blokken die, vergelijkbaar met puzzelstukken, alleen in elkaar passen als de combinatie ervan syntactisch correct is. Zie bijvoorbeeld Figuur 4, daar is te zien hoe de variabele in de boolean past, welke weer in het “als ... dan”-blok past. Dergelijke keuzes in het

ontwerp van de talen helpt voor beide de mate van mentale moeite te verminderen.



Figuur 4: De variabele past in de boolean, die weer in de “als ... dan” past

Scratch maakt gebruik van blokken die een boven- en onderkant hebben, zoals het “herhaal ...”-blok in Figuur 5a. Hierdoor is het duidelijk welke code wel en niet binnen de herhaling valt. Hetzelfde geldt voor conditionele blokken.

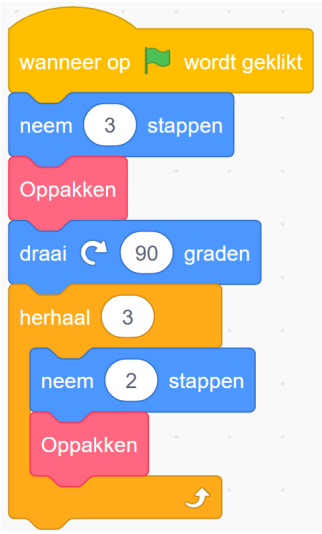
Scottie Go! heeft ook zijn eigen manier om code in loops en conditionele statements duidelijk te scheiden van code erbuiten. Blokken die tussen “REPEAT” en “END IF/REPEAT” toegevoegd worden, zijn ingesprongen doordat de blokken daartussen iets naar rechts liggen door hun pasvorm, zie Figuur 5b. Ook hierdoor wordt het duidelijker wat binnen en wat buiten de loop valt. De speler moet nog wel goed nadenken over hetgeen dat hij tussen die blokken neerzet. Hier moet goed worden nagedacht over welke blokken herhaald moeten worden en hoe vaak. Zeker wanneer dit aantal blokken groter wordt of als geneste herhaal-lussen gebruikt worden wordt dit een mentaal moeilijke operatie, maar voor Scratch geldt hetzelfde. Relatief verschillen de talen hierin dus niet. Het verschil met Scratch is dat de speler in Scottie Go! voor een “REPEAT”-blok of een “IF”-blok gebruik moet maken van een “END IF/REPEAT”-blok, terwijl dit bij Scratch niet nodig is. Dit zorgt voor iets meer mental operations, omdat de speler dat niet moet vergeten.

Al met al heeft Scottie Go! dus iets meer hard mental operations, maar het verschil is beperkt.

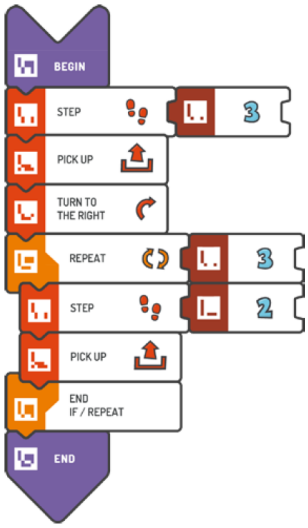
5.5 Premature Commitment

Wanneer de gebruiker een beslissing moet nemen terwijl nog niet alle benodigde informatie beschikbaar is, wordt gesproken over premature commitment [GP96]. Een simpel voorbeeld hiervan is wanneer je gedwongen zou worden te kiezen tussen een lepel of een vork, voordat je weet of je soep of aardappelen te eten krijgt. Dit probleem komt bij Scratch en Scottie Go! niet veel voor. Voor Scratch moet er alleen van tevoren bedacht worden wat voor type herhaalblok er gebruikt moet worden, zie Figuur 3, en of er een “als ... dan” of een “als ... dan - anders” blok gebruikt moet worden. In Scottie Go! moet er van tevoren gekozen worden of er een “REPEAT” of een “REPEAT WHILE” blok gekozen moet worden, maar voor conditionele statements hoeft die keuze niet van te voren gemaakt te worden.

Scratch vereist dus iets meer premature commitment, maar net als met hard mental operations is het verschil tussen de twee beperkt.



(a) De twee blokken binnen de lus zijn omsloten door het "herhaal ..."-blok



(b) De twee blokken binnen de "REPEAT"-lus zijn ingesprongen

Figuur 5: Vorm van herhaalblokken in Scratch en Scottie Go!

5.6 Secondary Notation

Secondary notation betreft onderdelen van een programmeertaal of editor die informatie toevoegen, maar niet deel zijn van de code zelf [GP96]. Dit kan bijvoorbeeld de opmaak van code zijn, of de mogelijkheid om comments toe te voegen.

In Scratch kunnen stukken code vrij geplaatst worden waar de gebruiker dat wil, de blokken die bij elkaar horen moeten natuurlijk wel aan elkaar vast zitten, maar de positie van stukken aparte code maakt niet uit. Hierdoor kan de gebruiker zijn code zo neerzetten dat het duidelijker wordt wat bij elkaar hoort, bijvoorbeeld door gerelateerde code dicht bij elkaar te zetten. Zie bijvoorbeeld Figuur 6, waar de code voor het bewegen van een sprite staat. In dat figuur is te zien dat de losse stukken code in dezelfde positie ten opzichte van elkaar staan als de pijltjestoetsen van een toetsenbord.

In Scottie Go! is het net als bij Scratch mogelijk om de code op te delen in functies en deze vrij neer te leggen. Het verschil is dat Scottie Go! maximaal drie extra functies ondersteunt, dus de mogelijkheden zijn beperkt. In Scratch is het ook mogelijk om comments toe te voegen, bij Scottie Go! is dit niet het geval. Scratch heeft dus meer mogelijkheden wat betreft secondary notation.

5.7 Visibility

Visibility bepaalt hoe makkelijk het is om de onderdelen en code van een programma te zien [GP96]. Voor Scottie Go! is de visibility zeer hoog, want alle code is tegelijkertijd zichtbaar op het bord waarop het neergelegd wordt. In Scratch is niet altijd alle code direct zichtbaar. Om de code die voor een bepaalde sprite geschreven is te bekijken, moet de gebruiker naar die sprite navigeren. Dit betekent dus ook dat alleen de code van de huidige sprite te zien is. Daarnaast is de code die voor een bepaalde sprite geschreven is niet altijd helemaal zichtbaar wanneer het programma groter is.

Figuur 6: Scratch code om een sprite te bewegen

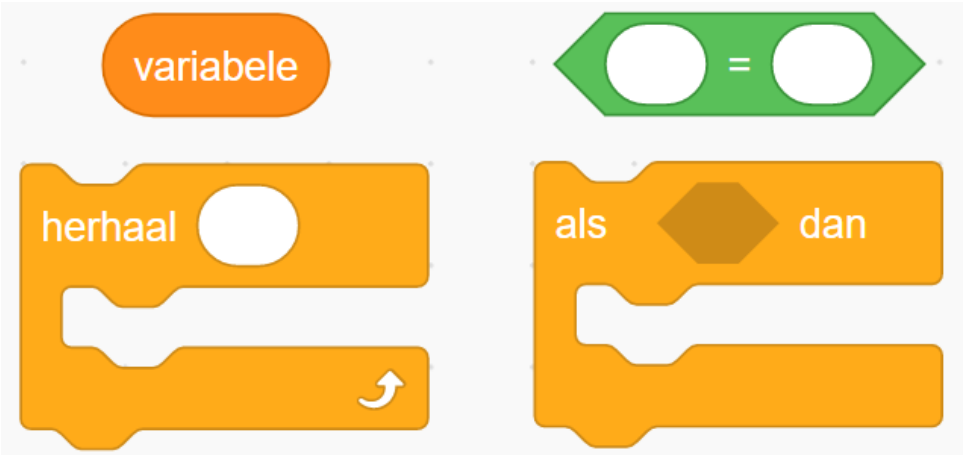
Om alles te zien zal de gebruiker dan moeten scrollen, dit kan zowel over de x-as zijn als over de y-as zijn. De visibility van Scratch is dus beduidend lager dan dat van Scottie Go!

5.8 Error-Proneness

Error-proneness is de vatbaarheid voor fouten [BG03]. Een fout tijdens het programeren zou bijvoorbeeld kunnen zijn dat een string, een variabele die een stuk tekst bevat, op een plek gebruikt wordt waar een getal hoort te staan, zoals in een optelling. Een error is in deze context een fout in het gebruik van de taal, zoals het hiervoorgenoemde voorbeeld of het vergeten van een accolade of puntkomma in C. Een verkeerd geprogrammeerd programma, zoals bijvoorbeeld een loop die te vaak uitgevoerd wordt, valt hier dus niet onder. Zowel Scratch als Scottie Go! heeft manieren tegen error-proneness gevonden. Beide maken gebruik van de vorm die blokken hebben: door blokken te gebruiken die, vergelijkbaar met puzzelstukken, inhammen en uitsteeksels hebben is het duidelijk welke blokken wel en welke blokken niet onder elkaar kunnen staan. Dit maakt beide talen minder error-prone. Scratch heeft ook, weer door de vormgeving van blokken, duidelijk gemaakt welk type variabele gebruikt kan worden in een bepaald blok, zie Figuur 7. Het is wel mogelijk om in variabelen tekst te gebruiken en die vervolgens op een plek te gebruiken waar dat niet hoort, zoals een “herhaal”-blok. Scratch geeft hiervoor geen waarschuwing. Scottie Go! gebruikt alleen maar numerieke variabelen. Hierdoor kunnen dus geen fouten gemaakt worden zoals in het eerder gegeven voorbeeld. De error-proneness van Scottie Go! is dus lager.

5.9 Abstraction

Als het mogelijk is voor de gebruiker van een taal om iets toe te voegen of te veranderen aan de notatie van de taal is er sprake van abstraction [BG03]. De meest gebruikte manier van abstraction is de mogelijkheid om zelf functies te definiëren. In Scottie Go! is het mogelijk om functies te maken, maar het aantal functies dat gemaakt kan worden is gelimiteerd tot drie. Scratch heeft ook een manier om functies te maken, namelijk door eigen blokken te definiëren. Daarmee kunnen zo veel functies gemaakt worden als de gebruiker wil. Deze blokken werken maar voor één sprite, als de gebruiker hetzelfde blok in een andere sprite wilt gebruiken, moet dat blok daar weer gedefinieerd worden. Anders dan bij Scottie Go! kan Scratch ook parameters toevoegen, ook weer zoveel als de gebruiker wil. De abstraction van Scratch is dus hoger dan die van Scottie Go!



Figuur 7: In een “herhaal”-blok past alleen een numerieke variabele, in een “als ... dan”-blok alleen een boolean

5.10 Role-Expressiveness

Als de role-expressiveness van een taal of programmeeromgeving hoog is, is het voor de gebruiker makkelijk aan de hand van het uiterlijk van een onderdeel te achterhalen wat de rol ervan is [BG03]. Scratch en Scottie Go! maken de rol van hun blokken op dezelfde manier duidelijk: beide talen hebben hun blokken onderverdeeld in categorieën. Binnen die categorieën hebben blokken een vergelijkbare rol en dezelfde kleur, zoals te zien in Appendix A en Appendix B. De role-expressiveness van Scratch en Scottie Go! is dus gelijk.

5.11 Consistency

Een programmeertaal is consistent als vergelijkbare constructen vergelijkbare notaties hebben [BG03]. In Scratch en Scottie Go! komt de consistency naar voren door de vorm van de blokken. De Scratch blokken die soortgelijke constructies hebben zien er hetzelfde uit: ‘hHerhaal” en “als ... dan”-blokken hebben beide een boven- en een onderkant met ruimte ertussen voor andere blokken (zie ook Figuur 7), variabelen zijn allemaal ovaal, booleans zijn allemaal zeshoeken, enzovoorts. Ook blokken in Scottie Go! die soortgelijke constructies hebben zien er hetzelfde uit: “REPEAT” en “IF”- blokken schuiven de blokken eronder iets naar rechts, zie Figuur 5b en Appendix B.2, tegels die een variabele verwachten hebben een inham aan de rechterkant, zie Appendix B, enzovoorts. De consistency van de twee talen is dus gelijk.

5.12 Diffuseness

Diffuseness is de hoeveelheid tekst, blokken of ruimte die nodig is voor de code [BG03]. Zowel Scratch als Scottie Go! hebben vrij hoge diffuseness ten opzichte van tekstuele programmeertalen, omdat ze beide blokken gebruiken die veel ruimte innemen vergeleken met tekst. De twee hebben verder gelijke diffuseness, omdat de grootte van de blokken vergelijkbaar is en er voor vergelijkbare constructies even veel blokken nodig zijn in beide talen.

5.13 Provisionality

De mate waarin het mogelijk is om dingen uit te proberen of te schetsen wordt provisionality genoemd [BG03]. In Scratch is het mogelijk om losse stukken code uit te voeren, zonder het hele programma te runnen. Door op het bovenste blok van een stuk code te klikken wordt dit uitgevoerd. Daarnaast kunnen sprites makkelijk aangemaakt of gekopieerd worden, wat het mogelijk maakt om code te veranderen en te proberen zonder gelijk de code van de originele sprite aan te passen. Het is zelfs mogelijk om code aan te passen terwijl het programma uitgevoerd wordt waardoor de invloed van kleinere aanpassingen in de code makkelijk bekeken kunnen worden. In Scottie Go! is het ook mogelijk om dingen te proberen, maar het proces duurt langer. Als de speler een stukje code wilt uitproberen moet het eerst ingescand worden en moeten alle animaties afgekeken worden. Scratch heeft door het aantal opties en de snelheid waarmee het aangepast kan worden een hogere provisionality.

5.14 Progressive Evaluation

Progressive evaluation is de mogelijkheid die een taal biedt om werk dat deels afgerond is te testen [BG03]. Dit lijkt erg op provisionality zoals hierboven beschreven, maar het verschil is dat het hier gaat om de mogelijkheid om een functie of programma dat nog niet compleet is te testen in tegenstelling tot de mogelijkheid om verschillende opties uit te proberen. In Scratch kan alle code die geschreven wordt gelijk uitgevoerd worden, zelfs wanneer een programma al draait kan er code toegevoegd worden die dan ook gelijk wordt uitgevoerd. In Scottie Go! is dit wat langzamer, omdat de code eerst ingescand moet worden en het niet tijdens het uitvoeren aangepast kan worden. Daarnaast moet de code die uitgevoerd wordt tussen een “BEGIN” en “END” blok staan, dus er zijn meer restricites. Een voordeel dat Scottie Go! wel heeft is dat het programma, of eigenlijk dus een level, terug wordt gezet naar de beginstaat als iets getest is. In Scratch is dit niet het geval, daar blijft de staat gelijk aan wat de eindstaat was. Als de gebruiker van Scratch iets wil testen, is het dus nodig om iets te programmeren om de beginstaat weer terug te krijgen. Deze voor- en nadelen van de twee talen zijn lastig naast elkaar te vergelijken omdat het per persoon zal verschillen waar de voorkeur ligt. Daarom beschouwen we de progressive evaluation van beide talen als vergelijkbaar.

Dimensie	Hoger bij
Closeness of Mapping	Scottie Go!
Viscosity	Scottie Go!
Hidden Dependencies	Scratch
Hard Mental Operations	Scottie Go!
Premature Commitment	Scratch
Secondary Notation	Scratch
Visibility	Scottie Go!
Error-Proneness	Scratch
Abstraction	Scratch
Role-Expressiveness	Gelijk
Consistency	Gelijk
Diffuseness	Gelijk
Provisionality	Scratch
Progressive Evaluation	Gelijk

Tabel 1: Cognitieve dimensies voor Scratch en Scottie Go!

6 Scratch of Scottie Go! om te Leren Programmeren

Scratch en Scottie Go! zijn beide ontwikkeld met het doel om kinderen te leren programmeren. Op een aantal vlakken lijken de twee talen op elkaar, bijvoorbeeld door het gebruik van blokken die op een bepaalde manier in elkaar passen, maar er is ook een aantal verschillen. In Sectie 5 zijn de twee talen vergeleken op het gebied van cognitieve dimensies. Hier gaan we dieper in op wat die verschillen betekenen voor het leerproces bij Scratch en Scottie Go!. Ook andere verschillen tussen de twee talen die niet onder cognitieve dimensies vallen zullen in deze sectie besproken worden. De focus bij deze vergelijking zal vooral liggen op de voor- en nadelen wat betreft leren programmeren.

6.1 Closeness of Mapping

Doordat de closeness of mapping voor Scottie Go! hoger ligt dan voor Scratch, is het voor de speler makkelijker om in te zien welke blokken van toepassing zijn op welke punten. Het voordeel dat Scottie Go! hierdoor heeft is dat het simpeler en duidelijker is, zeker voor beginners. Aan de andere kant geeft Scratch door de lagere closeness of mapping een beeld van programmeren dat dichter bij andere “echte” programmeertalen ligt, zoals de meeste tekstuele programmeertalen. Hierdoor zou de transfer, het toepassen van kennis en vaardigheden die zijn opgedaan in de ene situatie toepassen in een andere situatie, makkelijker kunnen verlopen. Dat gebeurt niet zomaar [PS92], maar als er begeleiding is voor de overgang kan dit een voordeel zijn als het uiteindelijke doel is om het kind te leren programmeren met een tekstuele programmeertaal.

6.2 Viscosity

Scratch heeft een hogere viscosity dan Scottie Go!, het is in Scratch dus meer werk om een kleine aanpassing aan de code te maken. Doordat dit proces lastiger is zal de gebruiker er meer over

moeten nadenken. Het werkgeheugen van de gebruiker, wat een beperkte grootte heeft, zal daardoor sneller vol raken. Dit heeft als gevolg dat het leren wordt beperkt omdat de gebruiker minder werkgeheugen over heeft daarvoor [KCR18a]. Daarnaast zal het werk trager gaan en zal de gebruiker sneller dingen vergeten of foutjes maken, wat tot frustratie kan leiden.

6.3 Hidden Dependencies

Hidden dependencies zijn, zeker voor beginnenende gebruikers, lastig om te doorgronden. Doordat Scottie Go! hidden dependencies totaal vermijdt, geeft het de speler één probleem minder om over na te denken. Scratch heeft wel hidden dependencies en is daardoor moeilijker te doorgronden voor beginners. Ook hier heeft Scottie Go! dus minder werkgeheugen van de speler nodig vergeleken met Scratch, waardoor er meer ruimte overblijft in het werkgeheugen om te leren.

6.4 Hard Mental Operations

Wat betreft hard mental operations bleek er maar een klein verschil te zijn tussen Scratch en Scottie Go!. Wanneer de speler van Scottie Go! een "REPEAT"-blok of een "IF"-blok gebruikt, moet ook een "END IF/REPEAT"-blok gebruikt worden. Bij Scratch zit dit automatisch in het blok verwerkt. Hoewel dit in Scottie Go! een kleine handeling is, neemt dit wel een deel van het werkgeheugen van de speler in beslag. Dit moet namelijk onthouden worden terwijl er aan de code tussen de blokken gewerkt wordt. Hierdoor heeft de gebruiker dus minder ruimte in het werkgeheugen voor het leren.

6.5 Premature Commitment

Ook de mate van premature commitment die nodig is in Scratch en Scottie Go! verschilt niet heel veel. Het verschil is dat in Scratch gekozen moet worden voor een "als ... dan" of een "als ... dan - anders"-blok wanneer een dergelijke constructie nodig is, terwijl dat in Scottie Go! niet van tevoren beslist hoeft te worden. Doordat de gebruiker in Scratch hier van tevoren over moet nadenken, wordt het werkgeheugen hierbij meer belast vergeleken met Scottie Go!. Bij Scratch is er dus minder ruimte over om te leren.

6.6 Secondary Notation

Scratch heeft meer opties voor secondary notation dan Scottie Go!. Neem bijvoorbeeld weer de code in Figuur 6: hier is het makkelijker te herkennen waarvoor deze code geschreven is. De code is in dezelfde vorm gelegd als de pijltjes op een toetsenbord. Daardoor is het niet nodig om ieder blok opnieuw te lezen en te begrijpen. In Scottie Go! is het beperkt mogelijk om code op een dergelijke manier neer te leggen. Naast bovenstaande heeft Scratch de mogelijkheid om comments toe te voegen aan de code. Met comments is het makkelijker om oude code terug te lezen en code van andere mensen te begrijpen. Het vergt wel werkgeheugen om tegelijkertijd te leren programmeren én comments voor de code bij te houden én na te denken over de positie van de code. Voor een beginnende gebruiker kan het werkgeheugen daardoor te vol raken om nog goed te kunnen leren. Het is wel zo dat Scratch deze opties niet opdringt, dus voor een beginner die niet weet dat de

optie bestaat zal het weinig invloed hebben. Er is meer onderzoek nodig om te bepalen of de eerst genoemde optie van secondary notation in Scratch voordelig is voor beginners.

6.7 Visibility

Scratch heeft minder visibility dan Scottie Go!, wat moeilijk kan zijn voor een beginner. Doordat er rekening gehouden moet worden met dingen die niet zichtbaar zijn, is er meer werkgeheugen nodig. Hierdoor blijft er minder ruimte over om te leren, dus voor een beginner is dit een groot nadeel. Aan de andere kant geldt ook weer dat dit wel eerder een probleem naar voren brengt wat programmeurs regelmatig hebben wanneer ze met meerdere files werken. Een gebruiker van Scratch is er daardoor al aan gewend als hij doorgaat met andere dingen programmeren. Zoals eerder genoemd is de transfer hiervan niet automatisch en is er begeleiding nodig om dit voordeel te bereiken.

6.8 Error-Proneness

Scottie Go! is minder error prone dan Scratch, vooral doordat het aantal opties gelimiteerder is, waardoor er dus minder fouten gemaakt kunnen worden. Zeker voor een beginner verlaagt dit het benodigde werkgeheugen, waardoor het leren beter gaat. Daarnaast zal de motivatie van de speler hoger blijven omdat het minder fouten maakt.

6.9 Abstraction

In het geval van abstraction heeft Scratch (met name in het begin) meer mogelijkheden dan Scottie Go!. Daardoor zal er in Scratch meer werkgeheugen nodig zijn, waardoor er minder ruimte overblijft om te leren. In Scottie Go! wordt abstraction pas later in het spel geïntroduceerd. Het lijkt aannemelijk dat de speler dan al enkele concepten heeft geautomatiseerd, waardoor er meer ruimte in het werkgeheugen is om rekening te leren houden met abstraction. Voor gebruikers die iets meer gevorderd zijn heeft Scratch het voordeel dat het meer mogelijkheden biedt vergeleken met Scottie Go!. Dit kan een voordeel zijn als daarna de stap zou worden gezet naar een tekstuele programmeertaal. Ook hier is het zo dat de transfer van die kennis wel begeleid moet worden en niet zomaar gaat. In het geval van abstraction heeft Scottie Go! dus voordelen voor een beginner, terwijl Scratch een voordeel heeft voor een gevorderde gebruiker.

6.10 Role-Expressiveness, Consistency en Diffuseness

De role-expressiveness, consistency en diffuseness van Scratch en Scottie Go! zijn vergelijkbaar, dus daar heeft geen van beide een voordeel vergeleken met de ander.

6.11 Provisionality

Scratch heeft een hogere provisionality dan Scottie Go!. Het biedt meer mogelijkheden om code tussendoor en snel uit te proberen. Hierdoor kan de gebruiker makkelijker combinaties van blokken proberen om te zien wat er gebeurt. Om code uit te proberen in Scottie Go! moet iedere keer een langer durende animatie afgekeken worden. Hierdoor zal een speler minder snel geneigd zijn om

even een stukje code te testen, aangezien het frustrerend kan zijn als er steeds gewacht moet worden op het resultaat. Als een speler minder vaak test, zoals in Scottie Go! naar verwachting het geval zal zijn, blijft de speler dus met meer code in het werkgeheugen zitten, wat nadelig kan zijn voor het leerproces.

6.12 Progressive Evaluation

Scratch en Scottie Go! hebben beide voor- en nadelelen wat betreft progressive evaluation. Scratch heeft het voordeel dat het testen van code sneller gaat, zoals ook benoemd onder Provisionality. Daarnaast is het niet nodig om een specifiek blok te bevestigen aan een functie die getest dient te worden, zoals Scottie Go! het "BEGIN" en "END" blok nodig heeft. Hierdoor is er voor een gebruiker van Scratch minder werkgeheugen nodig om een test uit te voeren. Aan de andere kant heeft Scottie Go! het voordeel dat het programma na een test wordt teruggezet naar de beginstand. Bij Scratch is dit niet het geval, daar blijven de sprites staan op de plek waar ze zijn als het programma stopgezet wordt. Dit zorgt ervoor dat de speler in Scottie Go! minder moeite hoeft te doen om later opnieuw het programma te testen. Hierdoor wordt er dus minder ruimte in het wergeheugen in beslag genomen wat dan weer gebruikt kan worden om te leren. Om te bepalen welke van deze voor- en nadelen zwaarder weegt is meer onderzoek nodig.

6.13 Conclusie cognitieve dimensies

Zoals hierboven blijkt, hebben Scratch en Scottie Go! beide voordelen en nadelen wat betreft de kenmerken die bij de cognitive dimensions vergeleken worden. Voor Scottie Go! ligt het voordeel vooral bij de eenvoud, waardoor het voor beginners makkelijker is om te begrijpen. Bij Scratch is het voordeel juist dat er meer mogelijk is, waardoor het meer lijkt op "echte" programmeertalen en de gebruiker dus beter kennis laat maken met de opties en problemen die daarbij komen kijken. Over het algemeen vereist Scottie Go! dus minder werkgeheugen van een speler, wat ervoor zorgt dat er meer ruimte overblijft om te leren. Scratch heeft een voordeel als het uiteindelijke doel is om van die taal naar een tekstuele taal over te stappen, omdat problemen die daar voorkomen ook in Scratch al aan bod komen. Het is dan wel zo dat de transfer van die kennis begeleid moet worden om ook daadwerkelijk nuttig te zijn tijdens die overstap.

6.14 Instructie

Het is ook nuttig om naar Scratch en Scottie Go! te kijken in een meer algemene zin, beide hebben namelijk een andere aanpak wat betreft het aanleren van programmeren. Scottie Go! geeft voor ieder nieuw element in het spel en voor ieder nieuw programmeerconcept uitleg aan de speler. Aan de andere kant geeft Scratch vrijwel geen uitleg. Wanneer een gebruiker voor het eerst een Scratch project aanmaakt, verschijnt er wel een tutorial die in het kort laat zien hoe Scratch werkt. Dit gaat niet verder dan de basisfunctionaliteiten van Scratch (het neerzetten van blokken en deze uitvoeren) en legt geen programmeerconcepten uit. Er zijn wel andere tutorials in Scratch aanwezig die uitgebreidere uitleg geven over Scratch en programmeren daarin, maar die zijn optioneel. Deze optie is niet duidelijk aanwezig en de verwachting is dat een kind deze niet snel zal vinden. Scratch werkt vanuit het principe van ontdekkend leren en gaat er vanuit dat de gebruiker zelf ontdekt hoe het moet en werkt. Scottie Go! is gebaseerd op het principe van directe instructie. Uit onderzoek

blijkt dat begeleiding en uitleg tijdens het leerproces helpt om de leerling de stof te laten begrijpen en onthouden [KSC06] [SJJ04]. Dit gebeurt bij Scottie Go! dus duidelijker en gerichter.

6.15 Feedback

Als laatste kijken we naar de feedback die gegeven wordt in Scratch en Scottie Go!. Het is voor het leerproces belangrijk dat er goede feedback wordt gegeven om de beheersing van de stof te vergroten en fouten om te zetten tot leermomenten [KCR18b]. In Scottie Go! wordt op verschillende manieren feedback gegeven. Ten eerste krijgt de speler feedback wanneer er een fout in de code zit; een error message laat zien wat er precies fout is en hoe dat op te lossen is. Ten tweede krijgt de speler tips wanneer de code wel werkt, maar het level daarmee niet opgelost wordt. De feedback hierbij is wel minder specifiek en is vaak alleen een algemene tip voor het level. Dit is waarschijnlijk geen punt omdat het meestal makkelijk te zien is waar de fout zit: de uitvoering van het programma stopt precies waar het fout gaat. Als laatste krijgt de speler een score tussen één en drie sterren wanneer het level gehaald is, waarbij de score hoger is als de code korter en daarmee beter is. De speler krijgt geen feedback over hoe zijn code korter kan. Een nadeel van het geven van feedback op deze laatste manier is dat de speler het niet hoeft te lezen. In Scratch is er helemaal geen feedback, omdat dat natuurlijk veel vrijer is en geen ingebouwde opdrachten heeft. Het feit dat Scottie Go! feedback geeft, pleit voor Scottie Go! ten opzichte van Scratch.

6.16 Conclusie

Scottie Go! heeft voor beginners veel voordelen tegenover Scratch. Het heeft directe uitleg door het hele spel heen, effectieve feedback wanneer er fouten gemaakt worden en het belast het werkgeheugen van de speler minder. Scratch heeft meer voordelen voor gevorderde gebruikers, zo heeft het meer vrijheid en mogelijkheden en sluit het beter aan bij tekstuele programmeertalen. Het lijkt er dus op dat Scottie Go! en Scratch elkaar aanvullen. Op basis van onze theoretische analyse kunnen we concluderen dat Scottie Go! beter is om de basis van programmeren aan te leren en Scratch meer geschikt is voor de gebruiker die die basis onder de knie heeft.

7 Vooronderzoek

Naast de theoretische analyse (zoals beschreven in de secties 5 en 6) is er een vooronderzoek voor een experiment gedaan, waarin onderzocht is in hoeverre het mogelijk is om Scratch en Scottie Go! in de praktijk te vergelijken. Voor dit vooronderzoek hebben wij de levels uit de eerste vier modules van Scottie Go! omgezet naar Scratch.

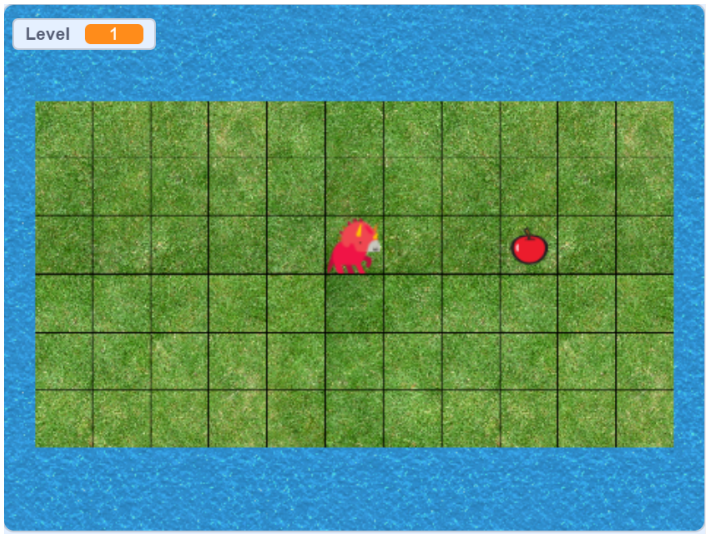
De mogelijke opzet van het onderzoek wordt beschreven in paragraaf 7.1. In de paragrafen 7.2 tot en met 7.5 wordt nader ingegaan op de vertaling. In paragraaf 7.6 wordt een aantal algemene verschillen genoemd tussen de de spellen in Scratch en Scottie Go!. Paragraaf 7.7 gaat in op de verschillen in de cognitieve dimensies tussen het spel in Scratch en Scratch in het algemeen. Als laatste wordt in paragraaf 7.8 beschreven hoe twee kinderen het Scratch spel hebben gespeeld en welke inzichten daaruit verkregen zijn.

7.1 Opzet

Het onderzoek dat hiermee uitgevoerd zou kunnen is als volgt: twee groepen kinderen spelen de vier modules, de ene groep in Scratch en de andere in Scottie Go!. De twee groepen worden onderverdeeld in groepjes van rond de drie kinderen, die met elkaar aan de slag gaan om de levels op te lossen. Na de vierde module volgt een toets die gelijk is voor beide groepen, waarbij de kennis die is opgedaan wordt getest. Door de scores op de tests van beide groepen met elkaar te vergelijken kan inzicht verkregen worden in bijvoorbeeld de vraag welk van de twee groepen het meest heeft geleerd.

7.2 Module 1

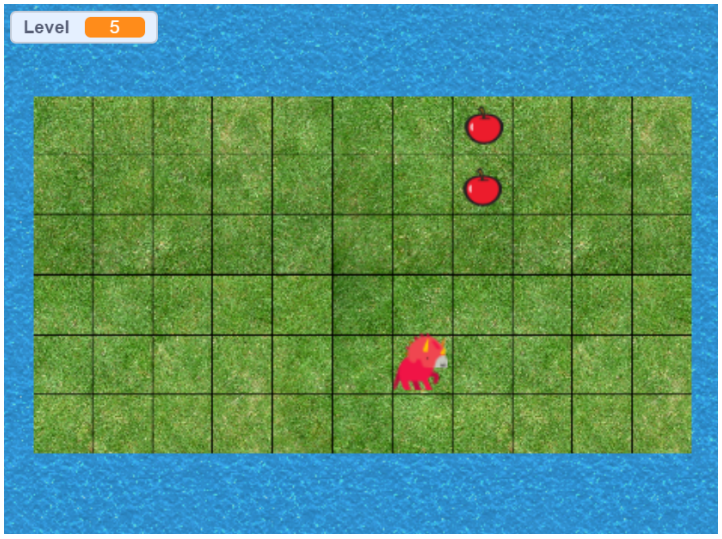
Scratch maakt voor de posities van sprites gebruik van een coördinatenstelsel. Met dit stelsel kunnen sprites zeer precies worden neergezet, maar dit veroorzaakt ook het eerste verschil met Scottie Go!. In Figuur 8 is het eerste level te zien. Om het level te halen moet Doris naar de appel lopen, in dit geval dus 3 vakjes. Een logische oplossing zou zijn om het blok "neem ... stappen" te gebruiken en "3" in te vullen op de plek van de variabele. Helaas verschuift Doris dan maar een klein stuk. Om bij de appel te komen moet er op de plek van de variabele "120" worden ingevuld: ieder vakje in het level is dus 40 stappen. Om het zoveel mogelijk op Scottie Go! te laten lijken is in de beschrijving van het eerste level van iedere module opgenomen dat elk vakje 40 stappen is. Dit vraagt dus om een extra rekenstap en een extra belasting van het werkgeheugen ten opzichte van Scottie Go!.



Figuur 8: Level 1 van module 1 in Scratch

Een soortgelijk verschil bestaat met het draaien van het personage. In Scottie Go! kan simpelweg gebruik worden gemaakt van een "TURN TO THE RIGHT" of "TURN TO THE LEFT" blok, terwijl het blok dat Scratch gebruikt, "draai ... graden", een getal in graden verwacht om de richting te bepalen. Hoewel dit iedere keer "90", "180" of "270" is, is het wel weer iets waar de speler extra over na moet denken. Als de speler dit al geleerd heeft bij rekenen en het al heeft geautomatiseerd zal dit verschil er niet zijn.

Vanaf level 5 zijn er meerdere plekken waar de speler heen moet lopen, zie Figuur 9. In Scottie Go! kan de speler over de twee plekken lopen, maar in Scratch werkt dit niet zo. Wanneer twee of meer blokken die de locatie of positie van een sprite aanpassen achter elkaar staan, wordt direct de eindlocatie uitgerekend en wordt de sprite daar neergezet. Als bijvoorbeeld twee "neem ... stappen"-blokken achter elkaar staan, de ene met "40" op de locatie van de variabele en de andere met "80", voert Scratch dit uit alsof er één blok met "120" staat. Hierdoor registreert Scratch niet dat er over beide appels wordt gelopen in level 5 en verder. Dit is op te lossen door een "wacht ... sec." blok tussen de twee blokken te plaatsen, zodat de sprite beide delen los uitvoert. Omdat de speler zelf de beweging van Doris programmeert, moet hij deze extra handeling ten opzichte van de speler in Scottie Go! zelf uitvoeren. Het helpt niet dat deze extra stap verwarrend kan zijn, omdat het onlogisch lijkt als al twee aparte 'neem ... stappen'-blokken worden gebruikt. Het zorgt opnieuw voor een extra belasting van het werkgeheugen.

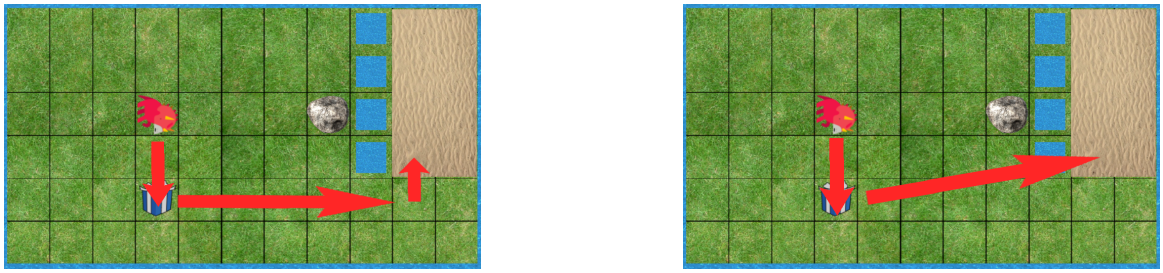


Figuur 9: Level 5 van module 1 in Scratch

7.3 Module 2

In deze module wordt het blok "PICK UP" geïntroduceerd in Scottie Go!. Zoals de naam al zegt kan de speler daarmee dingen oppakken in het level. Hiervoor is in Scratch geen blok, dus er is een eigen blok toegevoegd: "oppakken". In de definitie van dit blok is ook een "wacht ... sec."-blok toegevoegd, waardoor de speler er niet meer aan hoeft te denken om een "wacht ... sec."-blok te gebruiken.

Vanaf level 4 komen er regelmatig vakjes gevuld met water voor, zoals in Figuur 10 te zien is. Wanneer de speler hier overheen loopt, stopt het level en moet het opnieuw worden geprobeerd. De speler krijgt hierbij een bericht te zien dat dat aangeeft. Om dergelijke valstrikken te implementeren is het in Scratch gebruikelijk om het blok "raak ik ... ?" te gebruiken, waarbij gekozen kan worden voor de sprite die de speler gebruikt. Zie Figuur 11 voor een simpel voorbeeld hiervan. Wanneer de speler Doris laat lopen over het veld, zal het bijna altijd zo zijn dat Doris verplaatst wordt van de ene kant van het water naar de andere kant met een "neem ... stappen"-blok. Zoals eerder is beschreven, wordt Doris er dan direct overheen geplaatst en zal ze het water niet raken. Daarom



(a) Logische uitvoering (b) Echte uitvoering

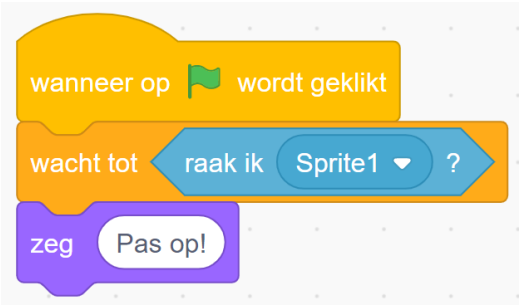
Figuur 10: Level 10 van module 2

is het niet mogelijk om alleen een “raak ik ...?”-blok te gebruiken. Om dat op te lossen houdt de sprite van Doris een variabele bij voor zijn oude en nieuwe positie, zowel voor de x-coördinaat als de y-coördinaat. Deze worden iedere 0,1 seconden geupdate. Ondertussen checkt ieder stuk water constant of Doris eroverheen is bewogen door de nieuwe en oude waardes van de positievariabelen te vergelijken. Als dit het geval is wordt het level stopgezet en wordt Doris naar het stuk water verplaatst om duidelijk te maken waar het fout ging. De speler kan vervolgens het level opnieuw proberen.

Vanaf level 4 kan de speler gebruik maken van het blok “spring”. Dit is ook een blok dat niet standaard in Scratch aanwezig is, maar door ons is toegevoegd. Met dit blok kan de speler over één vakje met water springen. De uitleg over dit blok wordt pas in level 5 gegeven, ondanks dat het wel al beschikbaar en nuttig is in level 4. Dit gebeurt in Scottie Go! ook op dat moment, dus dit is aangehouden.

Vanaf level 8 staan er af en toe stenen op het speelveld, ook te zien in Figuur 10. De speler kan niet door deze stenen heen bewegen en er ook niet overheen springen. De speler hoeft niet opnieuw te beginnen als hij dit wel probeert, net als in Scottie Go!. Wanneer de speler het toch probeert wordt Doris neergezet op het vakje waar ze probeerde over de steen te bewegen en wordt de rest van haar code verder uitgevoerd. Net als bij het water werken deze stenen door constant te checken of de speler er niet overheen is bewogen door de oude en nieuwe positievariabelen te vergelijken.

Door water en stenen op deze manier te gebruiken kunnen er situaties ontstaan waar het spel niet logisch werkt. Zoals eerder genoemd in Sectie 7.2 wordt een sprite direct op de berekende eindpositie geplaatst, in plaats van dat zij over het veld naar die positie beweegt. Hoewel het onlogisch lijkt, is het mogelijk dat Doris, buiten het zicht van de speler, tijdens het verplaatsen water raakt. Daardoor kan de speler ten onrechte af zijn. In Figuur 10 staat dit gevisualiseerd. Nadat Doris het cadeau heeft gepakt, zou je verwachten dat ze volgens de route uit figuur 10a naar het zand loopt. In Scratch gebeurt dit in de praktijk echter niet. Doris neemt daar de snelste weg, zoals eerder is beschreven. Dit is te zien in figuur 10b. Ook dit kan weer alleen opgelost worden door een “wacht ... sec.”-blok te plaatsen op de locatie waar de speler van richting verandert. Dit is dus weer een onnatuurlijke extra stap voor de speler, die meer werkgeheugen vergt dan eigenlijk nodig zou moeten zijn.



Figuur 11: Een simpel voorbeeld voor het blok “raak ik ... ?”

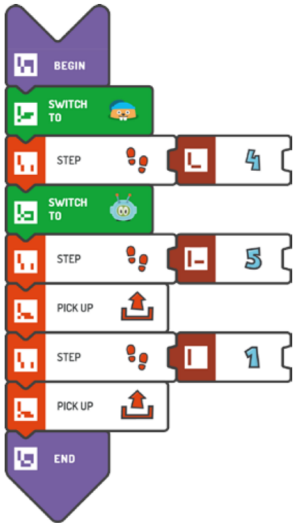
7.4 Module 3

In module 3 wordt het gebruik van het “herhaal ...”-blok uitgelegd. Deze gehele module is gebouwd rondom dit blok en het begrijpen daarvan. Om dit relatief moeilijke concept duidelijk te maken aan de speler is er in ons Scratch programma voor het begin van het eerste level een extra “level” toegevoegd. Hierbij wordt uitleg gegeven over het “herhaal ...”-blok. Daarnaast zijn de blokken die gebruikt moeten worden om het level te halen al in de programmeeromgeving geplaatst. De speler hoeft dan alleen nog maar de blokken in de juiste volgorde te zetten en de correcte getallen in te vullen in de blokken die een variabele nodig hebben.

Hoewel het gebruik van het “herhaal ...”-blok in ieder level aangemoedigd wordt, wordt dit niet geforceerd. Er is geen mogelijkheid om te controleren welke blokken gebruikt worden en daardoor kan er, anders dan in Scottie Go!, geen opmerking of score aan verbonden worden als de speler geen herhaling gebruikt.

7.5 Module 4

Module 4 van Scottie Go! introduceert een nieuw karakter in het spel: de bever. In Scratch is een vergelijkbaar karakter toegevoegd, namelijk de schildpad. De schildpad kan over het water bewegen en werkt als een brug voor Doris: wanneer de schildpad zich op een vakje met water bevindt kan Doris daar overheen bewegen zonder te springen. De schildpad kan ook door de speler bestuurd worden vanaf level 4. Op dit punt ontstaat er een groot verschil tussen Scratch en Scottie Go!. In Scottie Go! kan de speler eenvoudig wisselen tussen Scottie en de bever, zoals te zien is in Figuur 12. Dit kan in Scratch niet. Sterker nog, Doris en de schildpad zijn elk een Sprite, die apart geprogrammeerd moeten worden. Daar komt bij dat de code voor beide sprites niet tegelijk zichtbaar is. De speler moet er actief tussen wisselen. Daarentegen wordt de code voor beide sprites wel tegelijkertijd uitgevoerd in plaats van sequentieel, zoals in Scottie Go!. Dit alles bij elkaar maakt dat het spel in Scratch vanaf level 4 van module 4 beduidend meer belasting van het werkgeheugen vraagt dan het spel Scottie Go! zelf.



Figuur 12: Code om eerst de bever en daarna Scottie te bewegen

Na dit level worden de aanpassingen steeds ingewikkelder wat de speelbaarheid niet ten goede komt. Vanaf level 5 is het de bedoeling dat Doris mee kan reizen op de schildpad. Dit bleek lastig te implementeren in Scratch. Het lijkt erop dat sommige levels alleen werken als de speler precies het meest efficiente stuk code schrijft. Doet de speler dat niet, dan kunnen er onverwachte dingen gebeuren: de schildpad gaat eerder bewegen of Doris blijft achter. Het gegeven dat de speler zelf de beweging van de sprites programmeert, belet ons om het te verwerken in de bewegingsfuncties van een Sprite. Dat is een manier die normaal wordt gebruikt om dergelijke functionaliteiten te implementeren. De keuzes die zijn gemaakt bij het implementeren van het water spelen waarschijnlijk ook een rol. Naast het feit dat deze problemen de speelbaarheid in negatieve zin beïnvloeden maken ze dat de vergelijking tussen Scratch en Scottie Go! steeds meer scheef gaat. Er is daarom besloten om niet verder te gaan dan level 5 van module 4.

7.6 Algemene Verschillen

In tegenstelling tot Scottie Go! wordt een level in Scratch niet direct teruggezet in de startpositie wanneer de speler het niet haalt. In Scratch blijft een programma standaard in de eindstaat staan nadat alle code is uitgevoerd. Voor het spel is de mogelijkheid om een level terug te zetten toegevoegd. De speler kan het level terugzetten door op de spatiebalk te drukken. Dit wordt uitgelegd in het eerste level van module 1 en wordt een aantal keer herhaald in verdere levels. Er is gekozen om het terugzetten handmatig door de speler te laten in doen, in tegenstelling tot een automatische implementatie. De meest logische plek om automatisch terugzetten uit te voeren is aan het eind van de code waarmee Doris wordt bewogen. Aangezien de speler dit zelf programmeert is dat niet mogelijk. Een automatische implementatie zou nog mogelijk zijn door bij te houden of Doris een bepaalde tijd niet heeft bewogen. Dit zou kunnen aangeven dat de code van de speler afgelopen is. Als het level dan niet gehaald is, kan het worden teruggezet. Het probleem daarbij is dat het mogelijk is dat de speler iets in zijn code heeft opgenomen waardoor Doris voor een tijdje niks doet, bijvoorbeeld door een “wacht ... seconden”- blok of door meerdere “oppakken”-blokken achter elkaar te zetten. Waarschijnlijk zal dit geen correcte code zijn voor het level. Maar als

Doris daardoor wordt teruggezet terwijl nog niet alle code is uitgevoerd is dat wel verwarrend en frustrerend voor de speler. Daarbij komt het feit dat het Scratch spel vergeleken met Scottie Go! weinig animaties heeft. Daardoor is het voor de speler van het Scratch spel lastiger om te zien waar in de code zijn fout zit. Als hij zelf het speelveld kan blijven bekijken nadat zijn code is afgelopen is het zoeken van fouten makkelijker dan wanneer Doris automatisch wordt teruggezet na een bepaalde tijd. Dit alles bij elkaar genomen heeft ons doen besluiten om de speler zelf het level terug te laten zetten. Het terugzetten van een level kan gedaan worden op ieder moment, bijvoorbeeld als de geschreven code het level niet haalt of als de speler in het water is gevallen. Ook wanneer de speler een klein stukje code test kan het level teruggezet worden in de originele positie. Hiermee kan de speler dus snel code testen, bijvoorbeeld om te kijken of een deel van het level gehaald kan worden met de geschreven code. Doordat het nu mogelijk is om een level terug te zetten naar de beginpositie is het testen in Scratch vergelijkbaar met het testen in Scottie Go!. Het is wel zo dat het in Scratch sneller gaat, omdat de animaties in Scottie Go! langer duren. Voor de speler van het Scratch spel zal dit een voordeel zijn, omdat hij minder lang hoeft te wachten op animaties voor een snelle test. Het terugzetten van een level kan in Scratch ook per ongeluk tijdens de uitvoering van code gedaan worden. Dit komt doordat de code die de speler heeft geschreven parallel loopt aan de code waarmee het level wordt gereset. In dat geval zal Doris nog doorgaan met het uitvoeren van de code nadat het level is teruggezet. Hierdoor kan er verwarring ontstaan, aangezien Doris dan bijvoorbeeld weer doorloopt vanuit de startpositie, terwijl de speler probeerde helemaal opnieuw te beginnen met het level. Dit kan verholpen worden door het blok “stop alle” te gebruiken aan het eind van de functie die het level reset. Helaas zorgde dit weer voor problemen met andere spelelementen, namelijk het water en stenen, die niet goed meer werkten als het level opnieuw werd geprobeerd. Een andere implementatie van die spelelementen had er mogelijk voor kunnen zorgen dat het terugzetten beter had kunnen werken.

In Scottie Go! is duidelijk te zien hoe Scottie over het speelveld loopt. De animaties maken duidelijk wat de code precies uitvoert en hoe dat het spel beïnvloedt. In Scratch wordt een sprite altijd naar de eindlocatie verplaatst wanneer er gebruikt wordt gemaakt van de blokken die de speler aangeleerd krijgt. Er zijn wel blokken in Scratch die een animatie hebben, zoals “schuif in ... sec naar x: ... y:...”. Er is voor gekozen om deze blokken niet te gebruiken, omdat die minder goed overeenkomen met de blokken die Scottie Go! gebruikt. In Scratch is het dus minder duidelijk te zien hoe een sprite beweegt vergeleken met Scottie Go!. Voor de speler is daardoor moeilijker te zien wat er onderweg met Doris gebeurt en waar het eventueel mis gaat. Neem bijvoorbeeld een speler die Doris 40 stappen wil laten nemen, 90 graden wil laten draaien en weer 40 stappen wil laten nemen. Als hij per ongeluk 180 graden invult in plaats van 90, zal Doris eerst 40 stappen vooruit lopen, vervolgens 180 graden draaien - dan staat ze op dezelfde positie op haar kop - en als laatste weer 40 stappen lopen, terug naar de startlocatie. De speler ziet dit verloop niet en ziet alleen de begin- en eindstaat: Doris staat eerst op de startplek en vervolgens op dezelfde plek op haar kop. Voor een beginner is het dan lastig om te achterhalen waar de fout is gemaakt, want er lijkt bijna niks te zijn gebeurd.

Aangezien het gehele programma geschreven is in Scratch en in dezelfde omgeving wordt uitgevoerd, kan de speler bij alle elementen van het spel. Dit betekent dus dat de speler per ongeluk of expres dingen kan aanpassen of breken in het spel. Om dit te voorkomen hebben we code zo veel mogelijk opzij verschoven en onder comments gezet. Zonder voorkennis van Scratch is het in principe niet mogelijk om vals te spelen. Wel kan het gebeuren dat een speler per ongeluk bij de code van het

spel komt en daar zonder opzet iets verandert.

7.7 Cognitieve Dimensies

Een aantal van de cognitieve dimensies die in Sectie 5 beschreven worden, zijn voor Scratch veranderd door het specifieke gebruik voor ons onderzoek.

Doordat een groot deel van het programma al geprogrammeerd is en de speler alleen nog maar de beweging van Doris hoeft te programmeren, zijn er geen hidden dependencies meer. Net als in Scottie Go! hoeft de speler geen rekening te houden met de wisselwerking tussen verschillende variabelen. Ook wanneer de schildpad wordt geïntroduceerd zijn er geen hidden dependencies. Aangezien voor beide maar één functie geschreven hoeft te worden door de speler zijn deze wel afhankelijk (dependent), maar zijn deze niet verborgen (hidden).

Ook de visibility is groter, doordat de speler maar één sprite tegelijkertijd nodig heeft. Hierdoor komt het op hetzelfde nivieau als Scottie Go!, waar ook alle benodigde code zichtbaar is op één plek. Vanaf module 4 wordt de visibility van het Scratch programma wel weer lager, daar wordt namelijk de schildpad geïntroduceerd. Aangezien dit een andere sprite is, is het niet meer zo dat alle code op dezelfde plek te vinden is.

Als laatste zijn provisionality en progressive evaluation hoger gemaakt. Doordat de speler op elk moment het spel terug kan zetten naar het begin van een level is het makkelijker om stukjes code te testen.

Scratch ligt dus door deze implementatie iets dichter bij Scottie Go! wat betreft hidden dependencies en visibility. Verder zijn de provisionality en progressive evaluation relatief hoger gemaakt door een nadeel dat Scratch had tegenover Scottie Go! weg te nemen. Zie ook Tabel 2

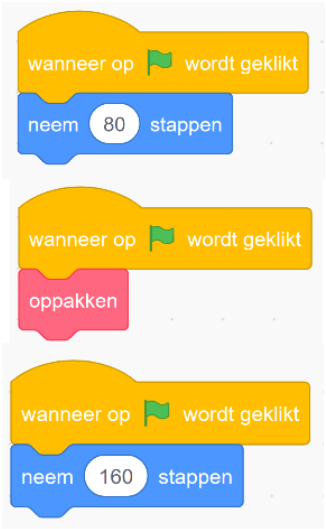
Dimensie	Verandering Scratch spel
Hidden Dependencies	Lager
Visibility	Hoger
Provisionality	Hoger
Progressive Evaluation	Hoger

Tabel 2: Veranderingen cognitieve dimensies in het Scratch spel vergeleken met Scratch in het algemeen

7.8 Tests

De vier modules zijn getest door twee kinderen om meer inzicht te krijgen in de werking ervan en de manier waarop kinderen de problemen oplossen. Beiden zitten in groep 7. Tester 1, een meisje, had van tevoren geen ervaring met programmeren, maar Tester 2, een jongen, heeft al eens eerder met Scratch gewerkt. In eerste instantie zijn de testers zelf aan de slag gegaan met de eerste module. Daarna hebben ze een enkele vraag beantwoord over die module. Omdat dit niet veel inzicht gaf in hun werkwijze, is er bij de overige modules via Skype meegekeken. Met tester 1 zijn module 2, 3 en 4 op Skype gedaan, met tester 2 alleen module 3 en 4 aangezien hij module 2 al zelf had gedaan.

(a) Alles onder elkaar



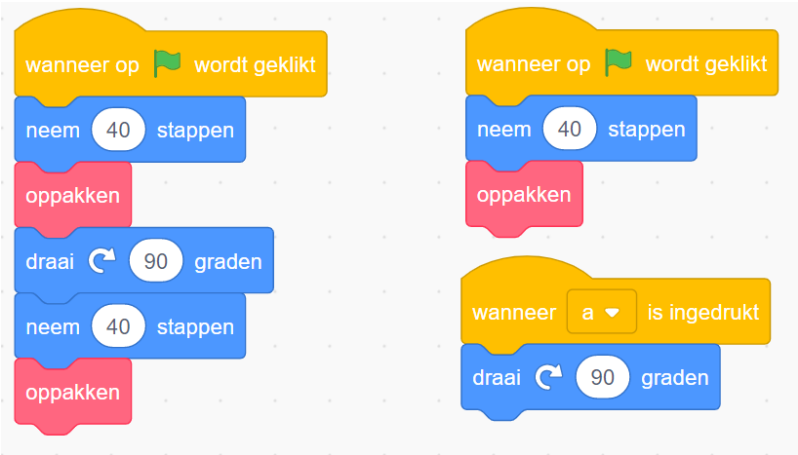
(b) Oplossing tester 1

Figuur 13: Oplossingen voor level 1 van module 2

Tester 1 begon module 2 met een interessante manier om de levels te halen: in plaats van de verwachte oplossing waarbij alle code achter elkaar staat, Figuur 13a, voerde zij steeds één blok tegelijk uit. Ze zette deze onder het "wanneer op [vlag] wordt geklikt"-blok en klikte iedere keer opnieuw op de groene vlag, zie Figuur 13b. Hierdoor was ze langer bezig met de levels dan eigenlijk nodig was en gebruikte ze Scratch niet op de manier waarop het bedoeld is. De eerste paar levels heeft ze op deze manier opgelost. Na uitleg over hoe de code werkt als meerdere dingen achter elkaar staan snapte ze het vrij snel en gebruikte ze het op die manier.

In de derde en vierde module haalde tester 1 de meeste levels door alle code onder elkaar te zetten. Bij de moeilijkere levels greep ze terug op haar strategie om het in meerder stappen op te lossen. Hoewel vooraf niet aan deze strategie gedacht is, kan het een manier zijn om een ingewikkelder level aan te pakken. Vanzelfsprekend is het daarbij van belang dat de speler na verloop van tijd deze strategie niet meer nodig heeft en levels haalt door alle code onder elkaar te zetten. Wat opviel was dat ze weining gebruik maakte van de functie om het level terug te zetten naar het startpunt. Meestal ging ze door met het level, ook wanneer het niet in één keer gehaald was. Ook dit was anders dan we van te voren hadden verwacht. Voor het leerproces is het van belang dat de speler leert om langere stukken code te schrijven. Voor een eventueel vervolgonderzoek is het dus aan te raden om hier een oplossing voor te bedenken. Dit zou kunnen door opnieuw te overwegen om levels wel automatisch terug te zetten. Een andere mogelijkheid is om gespeelde levels achteraf na te kijken en te beoordelen, zoals ook een spelling- of rekenopdracht wordt nagekeken.

Ook tester 2 loste de levels vaak op in meerdere kleine stappen, zowel in module 3 als in module 4. Soms was dit op dezelfde manier als tester 1, maar hij gebruikte ook een andere manier: hij maakte gebruik van "wanneer ... is ingedrukt", een blok dat niet benoemd was in de uitleg van het spel. Dit blok kende hij al door zijn eerdere kennismaking met Scratch. Hij gebruikte dit blok in combinatie met "draai ... graden". Daarmee koppelde hij het draaien van Doris los van de andere bewegingen, zie Figuur 14. Hiermee was hij nog een stap verder verwijderd van de oplossing die eigenlijk verwacht werd, maar het werkte voor hem erg goed.



Figuur 14: Links de verwachte code en rechts de code van tester 2

Deze manieren van werken geeft een verkeerde indruk van programmeren. Hoewel deze oplossingen in de context van het spel goed werken, is dit niet de manier van programmeren die we aan willen leren. Het zal voor de spelers lastig zijn om tegelijkertijd over te stappen naar "echt" programmeren en opnieuw te leren op welke manier code geschreven moet worden. Het opsplitsen van code om de levels op te lossen kan dus hun ontwikkeling als programmeur in de weg zitten.

Een laatste interessante observatie is dat tester 2 in module 3 vaak het "herhaal ..."-blok gebruikte op plekken waar dat eigenlijk niet nodig was. Hij deed dit, omdat in de opdracht stond dat het "herhaal ..."-blok gebruikt moest worden. Hij gebruikte bijvoorbeeld "herhaal [2]", "draai [45]" in plaats van een enkele "draai [90]". Het gebruik van herhalingen had dus beter uitgelegd kunnen worden.

8 Conclusie en Verder Onderzoek

Zoals blijkt uit de beschrijving van het vooronderzoek en de bijbehorende test van het Scratch programma in Sectie 7 is het niet veelbelovend om met de vertaling van Scottie Go! in Scratch het leerrendement van beide talen te vergelijken. In plaats van deze aanpak kan voor deze vergelijking gebruik worden gemaakt van de tutorials die Scratch heeft. Een nadeel hiervan is dat deze tutorials vrij kort zijn waardoor een concept beperkt geoefend wordt. Een betere mogelijkheid is om een eigen lesprogramma te ontwikkelen dat parallel loopt aan Scottie Go! wat betreft concepten die worden aangeleerd. Op deze manier is het ook mogelijk om dezelfde hoeveelheid tijd te geven om te oefenen met de stof. Zoals in Sectie 7.1 is beschreven kan vervolgens een experiment worden gedaan, waarin een groep Scottie Go! speelt en een andere groep het lesprogramma in Scratch volgt. Vervolgens kunnen beide groepen getoetst worden om te vergelijken hoeveel en wat ze geleerd hebben. Op deze manier is het mogelijk om op een praktische manier te onderzoeken of Scottie Go! inderdaad effectiever is dan Scratch om te leren programmeren, zoals wij concluderen op basis van onze theoretische vergelijking in Secties 5 en 6. Daaruit bleek dat Scottie Go! een betere manier van leren biedt voor de beginnende programmeur dan Scratch. Scottie Go! geeft directe uitleg en effectieve feedback aan de speler. Daarnaast is het minder belastend voor het werkgeheugen, waardoor de speler meer ruimte over heeft om te leren. Scratch heeft voordelen voor gevorderde gebruikers, zo

heeft het meer vrijheid en mogelijkheden en sluit het beter aan bij tekstuele programmeertalen. Op basis van dit onderzoek kan geconcludeerd worden dat Scratch en Scottie Go! niet met elkaar concurreren, maar elkaar waarschijnlijk aanvullen. Kinderen die willen leren programmeren kunnen de basis aanleren via Scottie Go! en daarna overstappen op Scratch. Het zal interessant zijn om verder te onderzoeken in hoeverre deze twee bij elkaar aansluiten.

Referenties

[PS92] David N. Perkins en Gavriel Salomon. "Transfer of Learning". In: *International Encyclopedia of Education, Second Edition*. Sep 1992.

[GP96] T.R.G. Green en M. Petre. "Usability Analysis of Visual Programming Environments". In: *Journal of Visual Languages & Computing* 7.2 (1996), p. 131–174.

[Pap99] Seymour Papert. *Eight Big Ideas Behind the Constructionist Learning Lab*. 1999. URL: http://stager.org/articles/8bigideas.pdf.

[BG03] Alan Blackwell en Thomas Green. "Notational Systems - The Cognitive Dimensions Framework". In: *HCI Models, Theories and Frameworks*. 2003, p. 103–134.

[SJJ04] Janine Swaak, Ton de Jong en Wouter R. van Joolingen. "The effects of discovery learning and expository instruction on the acquisition of definitional and intuitive knowledge". In: *Journal of Computer Assisted Learning* (mei 2004), p. 225–234.

[KSC06] Paul Kirschner, John Sweller en Richard Clark. "Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching". In: *Educational Psychologist* 41 (jun 2006). DOI: 10.1207/s15326985ep4102_1.

[Res+09] Mitchel Resnick e.a. "Scratch: Programming for All". In: *Communications of the ACM* 52.11 (2009), p. 60–67.

[Mal+10] John Maloney e.a. "The Scratch Programming Language and Environment". In: *ACM Transactions on Computing Education* 10.4 (2010).

[MAB13] Orni Meerbaum-Salant, Michal Armoni en Mordechai (Moti) Ben-Ari. "Learning Computer Science Concepts with Scratch". In: *Computer Science Education* 23.3 (2013), p. 239–264.

[KCR18a] Paul Kirschner, Kuce Claessen en Steven Raaijmakers. "De Leerkracht als Geheugenmanager". In: *Op de Schouders van Reuzen*. Dec 2018, p. 24–28.

[KCR18b] Paul Kirschner, Kuce Claessen en Steven Raaijmakers. "Effectieve Feedback". In: *Op de Schouders van Reuzen*. Dec 2018, p. 130–134.

[VT18] Maja Videnovik en Vladimir Trajkovik. "Using Scottie Go! as Game Based Learning Tool for Computational Thinking Course". In: *ICERI 2018 Proceedings* (nov 2018), p. 1337–1344. DOI: 10.21125/iceri.2018.1304.

[Zom19a] Peter Zomerdijk. *Lesplan Scottie Go! Educatie voor bovenbouw PO en onderbouw VO*. Versie 1.0. 3 dec 2019.

[Zom19b] Peter Zomerdijk. *Scottie Go! Docentenhandleiding*. 12 dec 2019.

[Cur] Curriculum.nu. *Bouwstenen - Curriculum.nu*. URL: https://www.curriculum.nu/bouwstenen/entry/377/ (bezocht op 24-11-2020).

[Goa] Scottie Go! *Scottie Go! Ontdek de wereld van het programmeren*. URL: https://scottiego.com/en/about-us/ (bezocht op 29-10-2020).

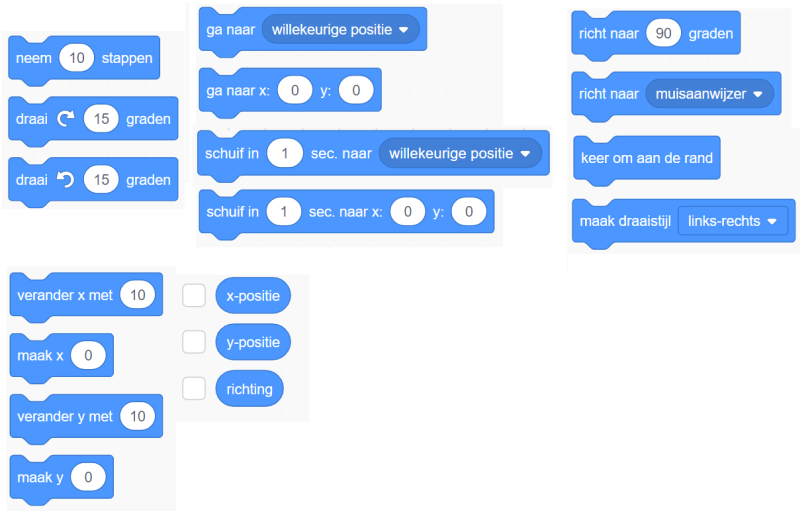
[Gob] Scottie Go! *Scottie Go! Ontdek de wereld van het programmeren*. URL: https://scottiego.nl/ (bezocht op 16-07-2020).

[Scr] Scratch. *Scratch - About*. URL: https://scratch.mit.edu/about (bezocht op 21-07-2020).

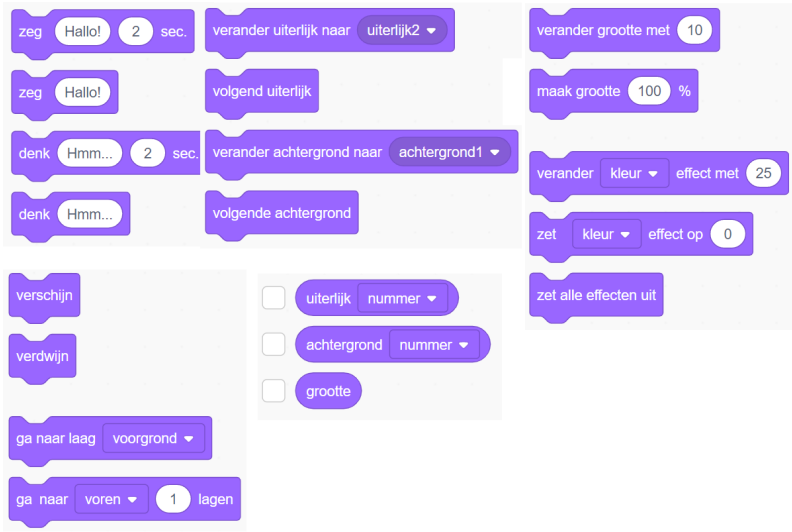
Appendices

A Scratch

A.1 Beweging



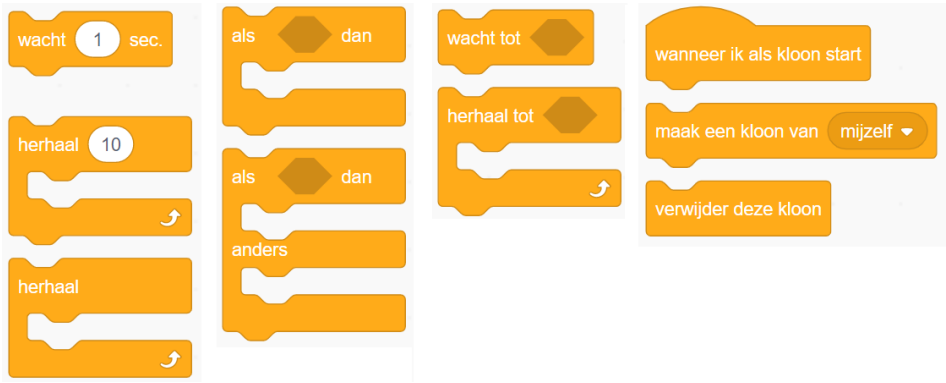
A.2 Uiterlijken



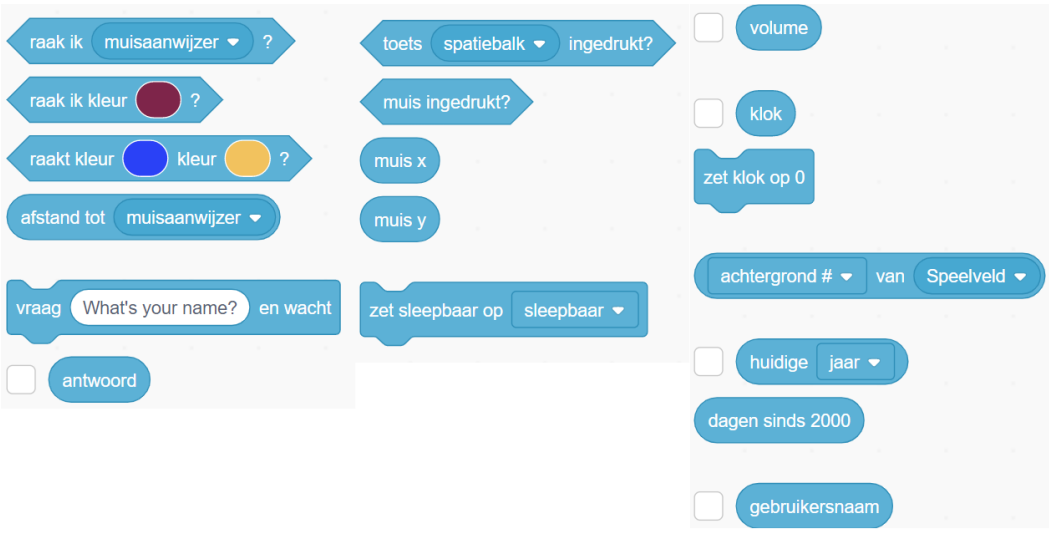
A.3 Geluid

A.4 Gebeurtenissen

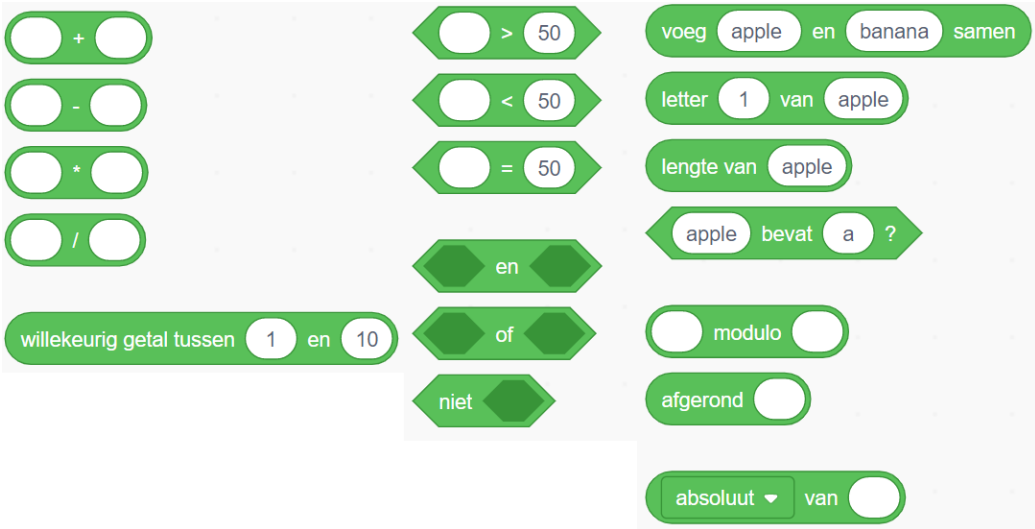
A.5 Besturen



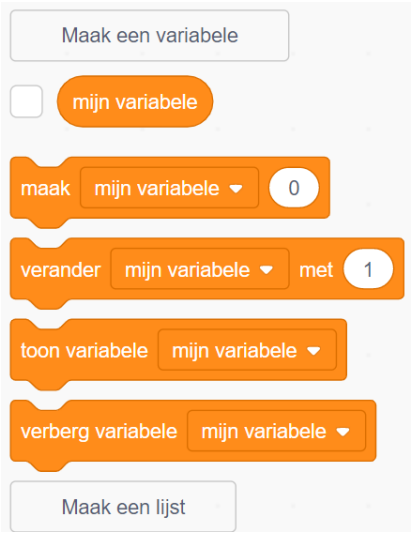
A.6 Waarnemen



A.7 Functies



A.8 Variabelen

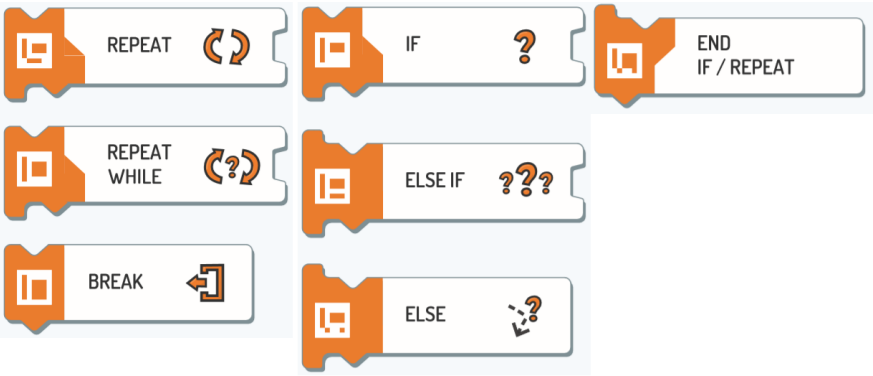


B Scottie Go!

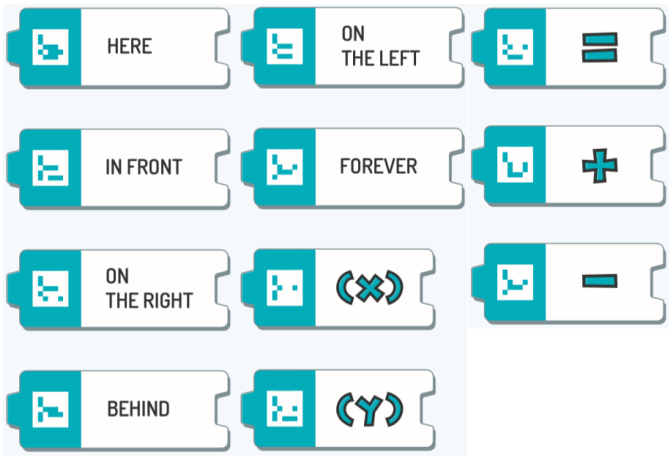
B.1 Begin and End

B.2 Action Tiles

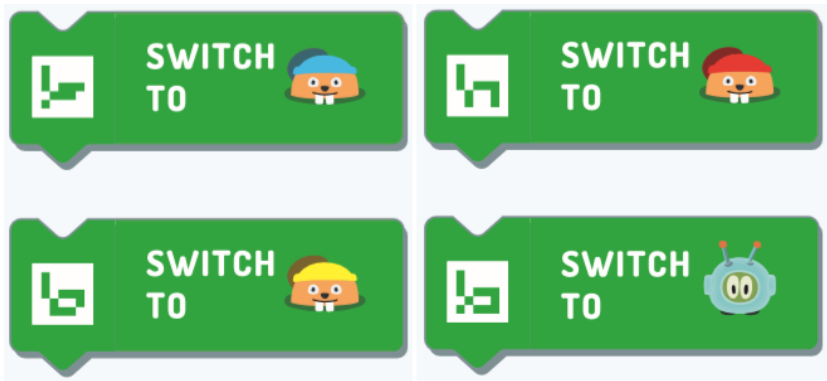
B.3 Loops and Conditional Statements



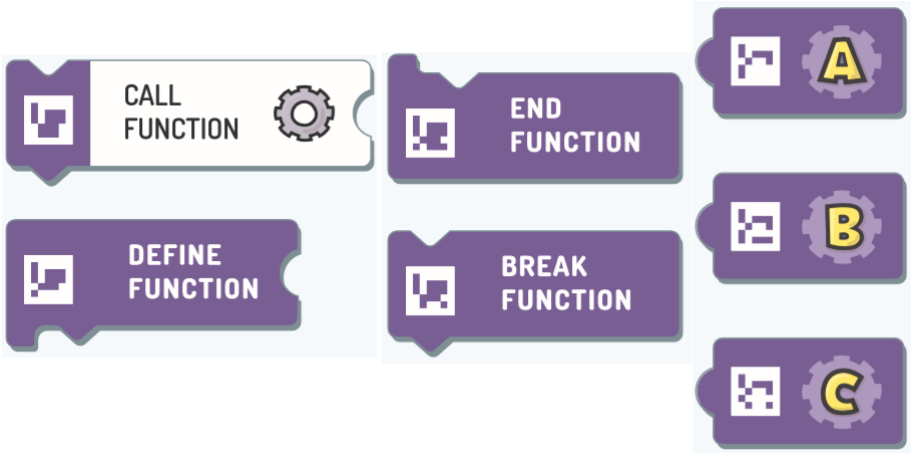
B.4 Logic



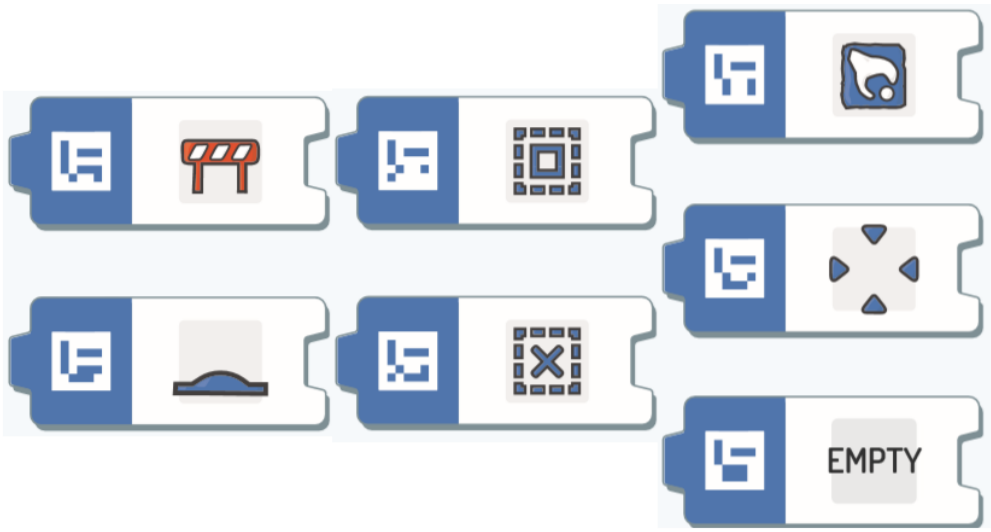
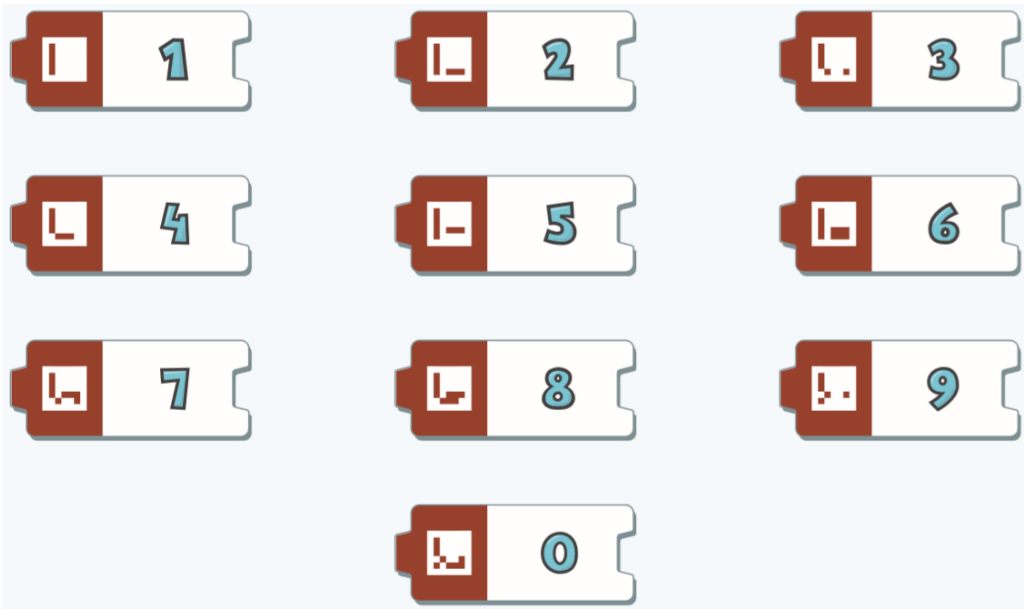
B.5 Switching Characters



B.6 Functions



B.7 Objects on the Screen



B.8 Digits