Reflectie op de app ‘Triddlers’

Bas Broere, Heleen Fritschy, Tosca Klijnsma & Lieke-Rosa Koetsier

Voorwoord

Deze reflectie is bedoeld om inzicht te geven in de app ‘Triddlers’: zowel hoe deze app werkt als waarom deze zo werkt. De app ‘Triddlers’ is gebouwd aan de hand van een opdracht voor het vak ‘Research & Devolpment’, een vak in de propedeuse informatica aan de Radboud Universiteit Nijmegen. Hopelijk kan deze app ook daarbuiten gebruikt worden: voor mensen die het leuk vinden om triddlers op te lossen (en vaardig genoeg zijn met een mobiele telefoon).

Dit document bestaat uit de volgende onderdelen:

Beschrijving

Inleiding

De app ‘Triddlers’ is een app waarmee je triddlers kan oplossen. Een triddler is vergelijkbaar met een Japanse puzzel, alleen heeft een triddler driehoekjes in plaats van vierkantjes en drie richtingen in plaats van twee. De puzzel bestaat uit een veld met driehoekjes en cijfers aan de rand van een rij of kolom. Het is de bedoeling om zo driehoekjes achter elkaar te kleuren dat het aantal gekleurde hokjes in iedere rij of kolom overeenkomt met de cijfers aan de rand. Een gebruiker van de app zal dus proberen een puzzel op te lossen, door bepaalde driehoekjes in te kleuren. Het leuke van een triddler is dat er een plaatje uit het veld met driehoekjes ontstaat op het moment dat je de puzzel opgelost hebt.

De app is als volgt opgebouwd: er is een beginscherm met de opties ‘Start met puzzelen’ en ‘Tutorial’. Als voor de optie ‘Tutorial’ gekozen wordt, volgen een aantal schermen met uitleg over de puzzel en een voorbeeldpuzzel die stap voor stap opgelost wordt. In zo’n scherm kan door de tekst gescrold worden.

Als men de optie ‘Start met puzzelen’ kiest, krijgt men een scherm te zien met drie lijsten met puzzels: kleine, middelgrote en grote puzzels. Binnen een lijst kan gescrold worden en een puzzel uitgezocht worden. Als men dan op een puzzel klikt, verschijnt deze puzzel in beeld en kan deze opgelost worden, door het selecteren van kleuren en driehoekjes. Er kan ook op de puzzel ingezoomd worden, zodat je de driehoekjes makkelijker kan kleuren. Men kan tijdens het oplossen van de puzzel teruggaan naar het scherm met de lijst met puzzels, de voortgang van de puzzel wordt toch opgeslagen. Als de puzzel af is, verdwijnen de cijfers aan de rand van de puzzel en is alleen nog het plaatje te zien. Wel kan een puzzel weer gereset worden en opnieuw opgelost worden. In het scherm met de lijsten met puzzels komt een sterretje voor de naam van een puzzel te staan als je deze puzzel opgelost hebt.

In figuur 1 zijn de vier verschillende soorten schermen die in de app voorkomen te zien. In figuur 2 is te zien hoe de puzzel eruitziet tijdens verschillende fases van het oplossen.

Figuur 1
De vier verschillende soorten schermen: het beginscherm, een scherm in de tutorial, het scherm met de lijsten met puzzels en het scherm waarin een puzzel opgelost kan worden.

Productverantwoording

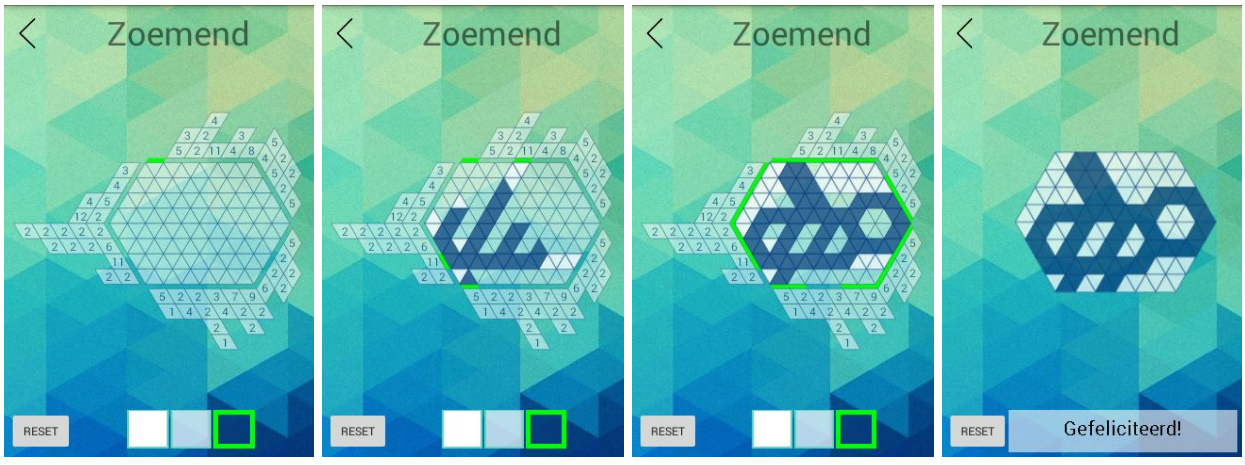
We kwamen op het idee van de triddlers via de site www.griddlers.net. Op deze site staan naast Japanse puzzels ook andere puzzels, zoals de triddler. De leden van onze projectgroep maken allemaal weleens een Japanse puzzel, daarom leek het ons leuk om een app voor zo'n soort puzzel te maken. Japanse puzzels zijn alleen vrij bekend en deze variatie met driehoekjes is minder bekend.

De meesten uit onze groep kenden de puzzel dan ook nog niet en zagen de triddler voor het eerst op de genoemde site. Al snel vonden we deze puzzel een leuke variatie op het maken van een Japanse puzzel. Omdat er nog geen app in de Google Play Store te vinden is waarin triddlers opgelost kunnen worden, leek het ons een goed idee om een app voor triddlers te maken. Er zijn namelijk al wel veel verschillende apps waarin Japanse puzzels opgelost kunnen worden. Het maken van deze app heeft dus een meerwaarde: het is immers de eerste.

Het idee van de puzzel hebben we dus van de site www.griddlers.net, echter hebben we van deze site geen enkele puzzel overgenomen. Alle plaatjes die bij de puzzels horen hebben we zelf getekend en omgezet naar een puzzel. Dit was redelijk veel werk, maar plezierig om te doen.

Wij denken dat de puzzel ook een meerwaarde heeft ten opzichte van een puzzelboekje of ten opzichte van de net genoemde site. Naast dat er voor zover wij weten geen in Nederland verkrijgbaar puzzelboekje bestaat met triddlers erin, heeft een app een aantal voordelen ten opzichte van een boekje. Zo kan in een app aangegeven worden wanneer een rij of kolom correct gekleurd is. Daarnaast wordt het plaatje vanzelf afgemaakt als de juiste hokjes donker gekleurd zijn, de andere hokjes hoeven niet meer handmatig wit gekleurd te worden. Ook is het mooi dat de cijfers wegvallen als de puzzel opgelost is.

Het voordeel van onze app ten opzichte van de genoemde site is met name de stijl. Wij vinden namelijk dat een puzzel op de site een vrij eenvoudige layout heeft, onze puzzel heeft wat meer kleur. Het voordeel van de site is dat deze een grotere voorraad aan puzzels heeft, maar wij zouden in de toekomst nog meer puzzels aan de app toe kunnen voegen.



Figuur 2
Het oplossen van een puzzel gaat door het selecteren van kleuren en driehoekjes (en het eventueel inzoomen van de puzzel). Als een bepaalde rij of kolom een correcte kleuring heeft, wordt deze rij of kolom aan de zijkant groen. Als de juiste driehoekjes donker gekleurd zijn, ontstaat een plaatje uit de kleuring en zijn de cijfers niet meer te zien.

Specificaties

Aan het begin van de ontwikkeling van de app hadden we een aantal eigenschappen die de app zou moeten hebben voor ogen. Een aantal eigenschappen daarvan bleken echter erg ambitieus, dus hadden we al snel besloten om deze eigenschappen niet waar te maken. De meeste specifieke eigenschappen die de app moet hebben, bleken echter pas tijdens de implementatie.

Hieronder laten we de meeste functionele eigenschappen die de app moet hebben - en waar als het goed is ook aan voldaan is - blijken uit een use case model. Daarna sommen we de functionele en niet-functionele eigenschappen op onder het volgende kopje.

Use case model

Een use case model beschrijft de actors die de app gebruiken en de verschillende use cases waarvoor de app gebruikt kan worden, ofwel waarvoor een actor de app gebruikt. Het doel van onze app is vrij duidelijk: het oplossen van een of meer puzzels. Daarom denken wij dat er ook maar een soort gebruiker is van de app: een gebruiker die een puzzel op wil lossen. De meest voor de hand liggende use case is dan ook 'een puzzel oplossen'. Een andere use case is 'uitleg krijgen over de puzzel', dus leren hoe in het algemeen een triddler opgelost kan worden.

Hieronder hebben we voor de twee verschillende use cases een happy path en een alternative path beschreven (op de wijze zoals dit in de cursus 'Object Oriëntatie' is uitgelegd).

Happy path voor use case 1: 'uitleg krijgen over de puzzel':

Pre: Tutorial en beginscherm werken naar behoren.

1. Het beginscherm van de puzzel geeft de opties 'Start met puzzelen' en 'Tutorial'.
2. Gebruiker kiest de optie 'Tutorial'.
3. Het eerste scherm van de tutorial komt in beeld met uitleg over de puzzel.
   Gebruiker kan de uitleg lezen en door de huidige pagina scrollen.
   Ook kan de gebruiker kiezen voor de opties 'Terug' en 'Volgende'.
4. De gebruiker leest de uitleg en kiest voor de optie 'Volgende'.
5. Gebruiker gaat door met de uitleg lezen en op 'Volgende' klikken tot en met de laatste pagina met uitleg.
6. Op de laatste pagina kan de gebruiker kiezen voor de opties 'Vorige' en 'Terug'.
7. De gebruiker kiest de optie 'Terug'.
8. Het beginscherm wordt opnieuw getoond.

Post: Gebruiker begrijpt hoe een puzzel opgelost kan worden.

Alternative path voor use case 1: 'uitleg krijgen over de puzzel':

Pre: Tutorial en beginscherm werken naar behoren.
   Gebruiker weet al grotendeels hoe een triddler opgelost moet worden.

1. Het beginscherm van de puzzel geeft de opties 'Start met puzzelen' en 'Tutorial'.
2. Gebruiker kiest de optie 'Tutorial'.
3. Gebruiker leest enkele pagina's van de uitleg en wil dan terug naar het beginscherm.

4. Gebruiker gebruikt de ingebouwde terugknop van zijn telefoon of tablet.
5. Het beginscherm wordt opnieuw getoond.

Post: Gebruiker begrijpt hoe een puzzel opgelost kan worden.

Happy path voor use case 2: ‘een puzzel oplossen’:

Pre: Beginscherm, scherm met lijsten met puzzels en scherm met een puzzel werken naar behoren.
Gebruiker weet hoe hij een puzzel op kan lossen.

1. Het beginscherm van de puzzel geeft de opties ‘Start met puzzelen’ en ‘Tutorial’.
2. Gebruiker kiest de optie ‘Start met puzzelen’.
3. Het scherm met de lijsten met puzzels wordt getoond.
4. Gebruiker scrolt door de lijsten met puzzels en selecteert een puzzel.
5. Het scherm met de betreffende puzzel wordt getoond.
6. Gebruiker selecteert driehoekjes als hij beredeneerd heeft dat deze gekleurd moeten worden.
7. Gebruiker selecteert een andere kleur, bijvoorbeeld wit, als hij beredeneerd heeft dat bepaalde driehoekjes juist niet gekleurd moeten worden.
8. Gebruiker gaat door met driehoekjes en kleuren selecteren. Ook zoomt de gebruiker soms in zodat de puzzel groter is en daardoor beter te zien, en hij beter driehoekjes kan selecteren.
9. Op een gegeven moment selecteert de gebruiker het enige driehoekje dat nog donker gekleurd moet worden.
10. De oplossing van de puzzel wordt getoond. De cijfers naast de puzzel vallen weg.
11. De gebruiker bekijkt de oplossing en gaat terug naar het scherm met de lijsten met puzzels.
12. Dit scherm met de lijsten met puzzels wordt getoond.
    (Nu zou een andere puzzel via hetzelfde pad opgelost kunnen worden.)

Post: Gebruiker heeft een puzzel opgelost, opgeloste puzzel is opgeslagen in de app.

Alternative path voor use case 2: ‘een puzzel oplossen’:

Pre: Beginscherm, scherm met lijsten met puzzels en scherm met een puzzel werken naar behoren.
Gebruiker maakt een of meer fouten tijdens het oplossen en begint opnieuw.
Gebruiker verlaat de puzzel (of de hele app) voor de puzzel helemaal is opgelost.

1. Het beginscherm van de puzzel geeft de opties ‘Start met puzzelen’ en ‘Tutorial’.
2. Gebruiker kiest de optie ‘Start met puzzelen’.
3. Het scherm met de lijsten met puzzels wordt getoond.
4. Gebruiker scrolt door de lijsten met puzzels en selecteert een puzzel.
5. De gebruiker selecteert driehoekjes naar eigen inzicht.
6. De gebruiker selecteert een andere kleur als hij die kleur wil gebruiken.
7. Op een gegeven moment selecteert de gebruiker een driehoekje dat hij toch niet wil selecteren.
8. De gebruiker klikt nog een keer op het driehoekje zodat het gedeselecteerd is.
9. Gebruiker gaat verder met driehoekjes en kleuren selecteren.
10. Gebruiker weet niet meer hoe hij de puzzel verder kan oplossen. Hij denkt dat hij een grote fout heeft gemaakt.
11. Gebruiker selecteert de button ‘Reset’.
12. Puzzel is weer leeg zoals in het begin.
13. Gebruiker begint opnieuw met de puzzel oplossen.

14. Gebruiker gaat terug naar het scherm met de lijsten met puzzels, als de puzzel deels opgelost is.

Post: Gebruiker heeft een puzzel deels opgelost, de voortgang van de puzzel is opgeslagen in de app.

Ander paden zouden nog het draaien van de puzzel in beschouwing kunnen nemen (van potrait mode naar landscape mode of andersom). Ook zouden wij onderscheid kunnen maken tussen het gebruik van de app op een telefoon en op een tablet. We hebben echter geprobeerd de app zo te maken dat het gebruik op een tablet niet echt verschilt van het gebruik op een telefoon (behalve dat op de tablet de puzzels groter zijn en er daardoor minder hoeft te worden ingezoomd). Echter denken wij dat de meeste functionele eigenschappen al in deze scenario's verborgen zitten.

Eigenschappen

Hieronder is een lijst met noodzakelijke eigenschappen van de app. Deze lijst volgt grotendeels uit de use cases en de gegeven paden. Ook hebben we een lijst met overige eigenschappen opgesteld. Een aantal van deze eigenschappen blijkt ook al uit de verschillende paden.

Benodigde eigenschappen van de app:

- Er kan gewisseld worden tussen de verschillende soorten schermen.
- De tutorial is goed te lezen doordat er door de tekst gescrold kan worden.
- Er kan goed door de verschillende pagina's in de tutorial genavigeerd worden met de knoppen 'Vorige' en 'Volgende' (op de eerste en laatste pagina staat een 'Terug').
- Er kan door de lijsten met puzzels gescrold worden in het scherm waarin alle puzzels benoemd staan.
- De driehoekjes krijgen de juiste kleur bij het selecteren.
- Er kan gewisseld worden van kleur door het selecteren van een kleur.
- Als de telefoon of tablet gedraaid wordt, blijft hetzelfde scherm van de puzzel te zien (alleen dan in landscape mode of juist portrait mode), inclusief voortgang.
- De puzzel kan ingezoomd worden, zodat deze beter te zien is en de driehoekjes beter te selecteren zijn.
- Als je de puzzel hebt opgelost, is het ook duidelijk dat dit zo is, door bijvoorbeeld de tekst 'Gefeliciteerd!' onderaan in beeld.
- Als je de puzzel opgelost hebt, kun je daar niets meer (per ongeluk) aan veranderen, tot je op 'Reset' drukt.
- Als je een puzzel opnieuw wil maken of veel fouten hebt gemaakt, kun je met de knop 'Reset' een puzzel weer 'leeg' maken.
- Als een puzzel (deels) is opgelost, blijft de voortgang opgeslagen.
- Als je het laatste juiste donkere hokje selecteert, verschijnt de oplossing meteen in beeld. Zo hoef je niet nog alle overige hokjes wit te kleuren.

Overige eigenschappen van de app:

- Als je een vakje geselecteerd hebt, kan deze meteen gedeselecteerd worden door er nog eens op te klikken.
- Aan de rand van een rij of kolom komt een groen streepje te staan als deze rij of kolom een correcte kleuring heeft (het juiste aantal donker gekleurde hokjes na elkaar). Een rood streepje wordt getoond als het aantal donker gekleurde hokjes groter is dan de som van de cijfers aan de rand.
- Als je een puzzel opgelost hebt, wordt het plaatje getoond zonder de cijfers langs de rand.
- Als je een puzzel opgelost hebt, verschijnt er een sterretje voor de naam van de puzzel in het scherm met de lijsten met puzzels. Zo weet je daar al dat je met die puzzel klaar bent.

Ontwerp

Globaal ontwerp

Ons idee was het MVC-model duidelijk te implementeren voor het spel, maar we zijn iets anders te werk gegaan dan in het klassieke MVC-model.

We hebben gebruik gemaakt van twee views, één voor het spel zelf en één voor de kleurselectie. Deze hebben allebei hun eigen controller omdat ze specifieke, verschillende taken uitvoeren. Om de samenwerking tussen deze twee controllers te coördineren hebben we een derde controller geïntroduceerd. Deze main controller krijgt de informatie over de input van de views en geeft dit door aan de twee subcontrollers die de informatie weer doorgeven aan het model. Het model kan hier op reageren en dit via een van de subcontrollers weer doorgeven aan de main controller.

Het model bestaat natuurlijk uit het spel zelf en al zijn componenten, maar ook hierin hebben we een opsplitsing moeten maken in verschillende taken. Om goed onderscheid te kunnen maken tussen verschillende kleuren en richtingen hebben we enumeratietypen gemaakt. Dit maakte het geheel een stuk duidelijker en overzichtelijker.

Een voorbeeld: De user heeft de kleur blauw geselecteerd en klikt een hokje aan dat al blauw is gekleurd. De view van het spel geeft de coordinaten van het event door aan de main controller. Deze geeft de coordinaten weer door aan de spelcontroller. De controller vertaalt dit naar coordinaten in het spel en geeft dit door aan het model. Het model bekijkt welke kleur het hokje heeft en ziet dat de te geven kleur hetzelfde is als de huidige kleur. Het spel besluit daarom het hokje te ontkleuren. Deze wijziging wordt doorgegeven aan de spelcontroller die dit weer aan de view laat weten.

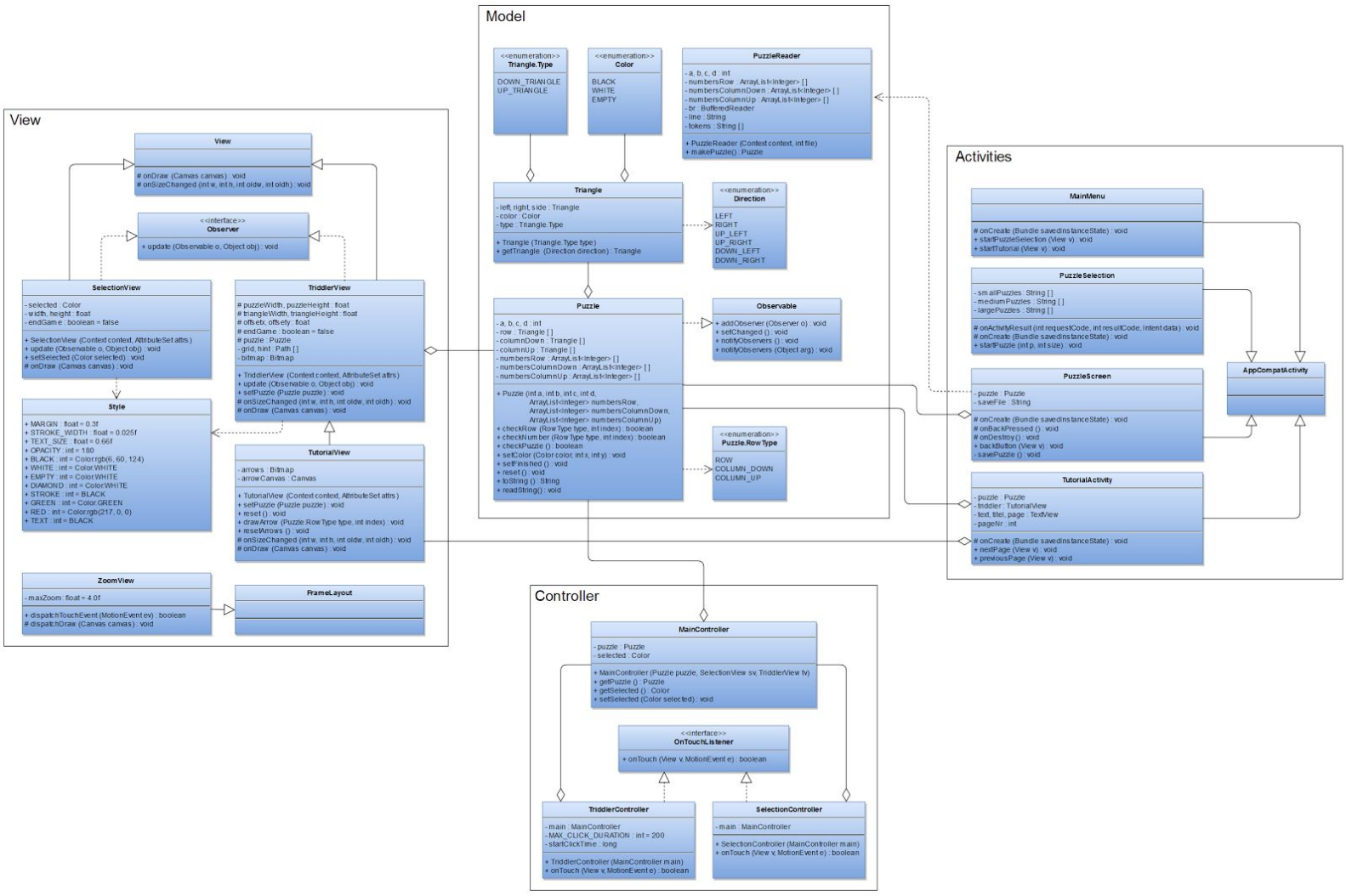
De app heeft naast het spel nog drie activiteiten: een hoofdmenu, een selectiemenu en een tutorial. Het hoofdmenu is een vrij eenvoudige activiteit waar niet veel anders in gebeurt dan het aanroepen van twee subactiviteiten, namelijk het selectiemenu en de tutorial.

Het selectiemenu is enigszins complexer gezien het ook gebruik maakt van het model. Deze activiteit controleert namelijk voor elke mogelijke puzzel of de user deze al heeft opgelost en geeft een opgeloste puzzel aan met een sterretje voor de naam van de puzzel. We zouden dit graag los hebben gehaald van het model en dit op een andere manier hebben geïmplementeerd, maar helaas is dat niet meer gelukt.

Vanuit het selectiemenu wordt een puzzelactiviteit aangemaakt. In deze activiteit zit alleen een main controller. Het eerder genoemde MVC-model regelt de rest.

De tutorial maakt gebruik van zijn eigen view en controller. Deze controller zit min of meer ingebakken in de tutorialactiviteit zelf. We hebben hiervoor gekozen omdat de tutorial activiteit op zichzelf een kleine activiteit is, waardoor een aparte controller maken niet nodig is.

Detailontwerp



Figuur 3
UML-klassendiagram. Hierbij zijn alleen de essentiële attributen en belangrijkste publieke methoden weergegeven. Getters en setters zijn voor de leesbaarheid ook weggelaten.

In het UML-diagram hierboven is de opdeling van de klassen in Model, View, Controller en Activities duidelijk te zien, zoals omschreven in het globaal ontwerp. Hieronder gaan we kort per module door alle klassen heen en omschrijven we de belangrijkste attributen en methodes per klasse. Alleen de belangrijkste klassen zullen we in detail uitwerken, van andere klassen geven we een korte omschrijving.

Model

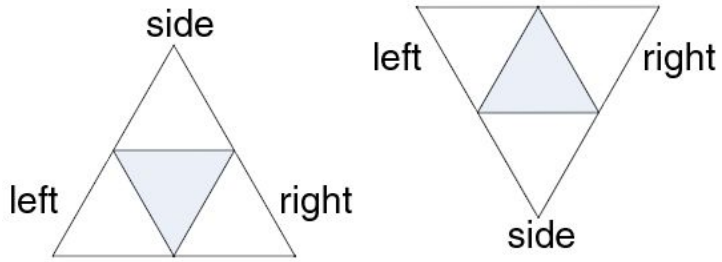
Het model is verantwoordelijk voor de logica en de interne representatie en status van de puzzel. De belangrijkste klasse in het model is dan ook de klasse *Puzzle*. De klasse *Triangle* wordt gebruikt in de klasse *Puzzle* om het grid te representeren. Verder hebben we nog een aantal enumeratietypes, zoals *Color* voor de verschillende kleuringen van een driehoek. *BLACK* betekent bijvoorbeeld dat een driehoek ingekleurd is, *WHITE* dat een driehoek wit gekleurd is en *EMPTY* dat een driehoek leeg is.

Het enumeratietype *Triangle.Type* wordt gebruikt om aan te geven of een driehoek met de punt naar boven wijst (*UP_TRIANGLE*) of naar beneden (*DOWN_TRIANGLE*). *Direction* geeft alle verschillende richtingen waarin in het grid gelopen kan worden, zoals RIGHT en DOWN_LEFT. *Puzzle.RowType* wordt gebruikt in Puzzle om gemakkelijk onderscheidt te maken tussen de drie verschillende rijen waarin nummers kunnen staan. Zo gebruiken we *ROW* voor de rijen in horizontale richting die van links naar rechts lopen, *COLUMN_DOWN* voor de kolommen die van rechtsboven naar linksonder lopen en *COLUMN_UP* voor de kolommen die van rechtsonder naar linksboven lopen.

Een andere klasse is de klasse *PuzzleReader*, die verantwoordelijk is voor het creëren van puzzels aan de hand van een textfile. Hieronder gaan we dieper in op de klasse *Triangle* en de klasse *Puzzle*.

Triangle

Elke Triangle bevat een kleur van type *Color*, een *Triangle.Type* en verwijzingen naar alle drie zijn buren *left*, *right* en *side*, mits deze bestaan. Hierbij verwijst *left* naar de linkerbuur, *right* naar de rechterbuur en *side* naar de onderbuur als het om een *UP_TRIANGLE* gaat of bovenbuur als het om een *DOWN_TRIANGLE* gaat.



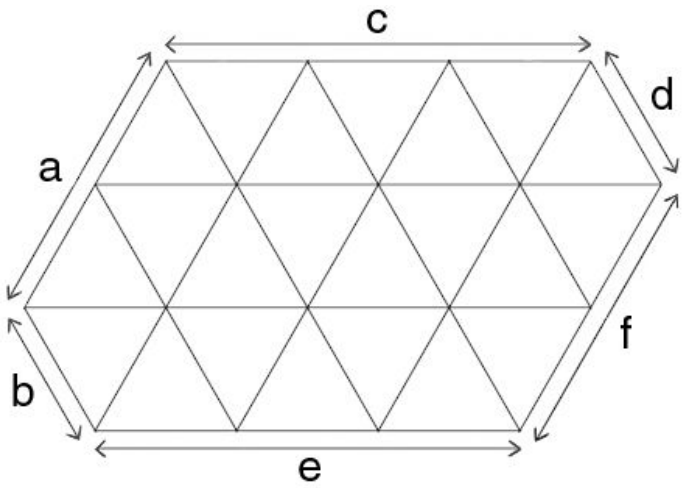
Figuur 4
Een DOWN_TRIANGLE (links) en een UP_TRIANGLE (rechts) met alledrie de buren.

De belangrijkste methode van *Triangle* is de methode *getTriangle (Direction direction)*, die van de huidige driehoek de aangrenzende driehoek geeft in de aangegeven richting. Roep je bijvoorbeeld *getTriangle (Direction.RIGHT)* aan dan zal je altijd de rechterbuur *right* van de huidige driehoek terugkrijgen. Roep je echter *getTriangle (Direction.DOWN_LEFT)* aan, dan zal je in het geval van een *DOWN_TRIANGLE* de linkerbuur *left* terugkrijgen en in geval van een *UP_TRIANGLE* de onderbuur *side*. Zo kun je gemakkelijk in het grid in een van de zes richtingen lopen zonder rekening te houden in welk type driehoek je je bevindt. Daarnaast zijn er nog de nodige getters en setters zoals *getType()*, *getColor()*, *setColor (Color color)*, *setLeft (Triangle triangle)*, *setRight (Triangle triangle)* en *setSide (Triangle triangle)*.

Puzzle

De klasse *Puzzle* bevat zijn dimensies, het grid van driehoeken en de nummers langs de rand van de puzzel. Een triddler heeft normaalgesproken een grid in de vorm van een zeshoek. Voor de dimensies van de puzzel slaan we vier waarden op: *a*, *b*, *c* en *d*. Waarbij *a* de lengte van de linker-bovenzijde is, *b* de lengte van de linker-onderzijde, *c* de lengte van de bovenzijde en *d* de lengte van de rechter-bovenzijde. De lengtes van de overige twee zijdes (*e* en *f*) zijn af te leiden uit de lengtes van *a*, *b*, *c*, en *d*.

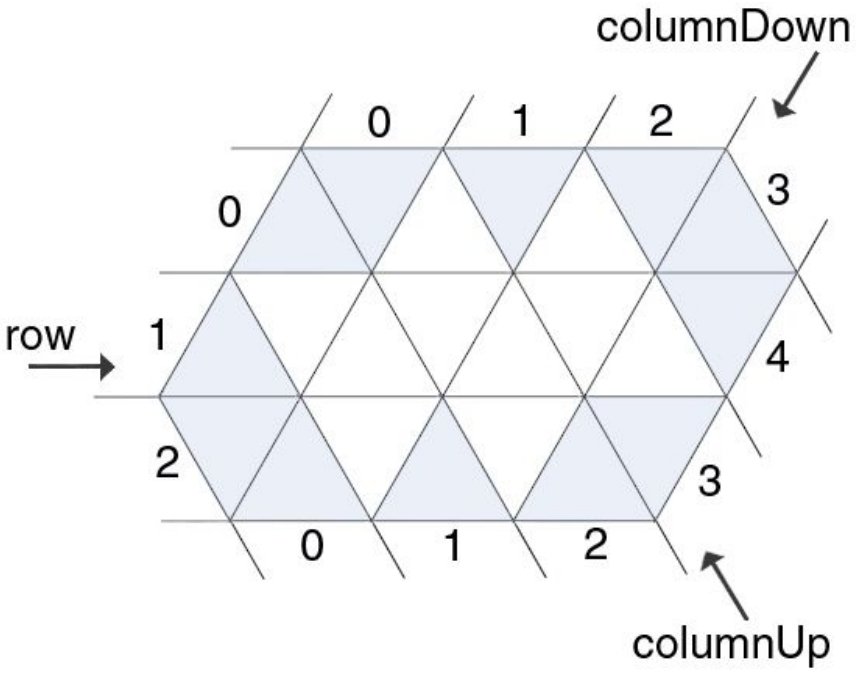
Het is mogelijk dat één of meerdere van deze waardes gelijk is aan nul, waardoor er in plaats van een zeshoek een vijfhoek, vierhoek of driehoek ontstaat. Aan de hand van deze dimensies maken we in de constructor van *Puzzle* een leeg grid aan, waarbij we elke driehoek correct linken aan zijn buren.



Figuur 5
Een standaard triddlergrid waarbij a = 2, b = 1, c = 3, d = 1, e = 3 en f = 2.

Voor het grid van driehoeken slaan we alleen verwijzingen op van de driehoeken aan de rand, en dat voor elk rijtype apart. Zo krijgen we drie arrays van *Triangles*: *row*, *columnDown* en *columnUp* van lengtes *a* + *b*, *c* + *d* en *e* + *f*, en kunnen we gemakkelijk het eerste driehoekje in een bepaalde rij of kolom opvragen. Met behulp van de methode *getTriangle (Direction direction)* in *Triangle* kunnen we dan over een rij of kolom lopen.

Het opslaan van de nummers langs de rand doen we tevens voor elk rijtype apart en hiervoor gebruiken we dezelfde indexering. Zo krijgen we *numbersRow*, *numbersColumnDown* en *numbersColumnUp*, waarbij voor elke rij of kolom alle nummers in een ArrayList worden opgeslagen. Het complete plaatje van een puzzel hoeft niet opgeslagen te worden. We gaan ervan uit dat al onze puzzels eenduidig oplosbaar zijn. Dus wanneer alle rijen in alle richtingen volgens de cijfers juist zijn ingevuld betekent dit dat de puzzel correct is opgelost.



Figuur 6
Alle driehoeken langs de rand die worden opgeslagen met hun rijtype en indexering.

De belangrijkste methodes van Puzzle zijn weergegeven in het UML-diagram. De methode *checkRow (RowType type, int index)* levert *true* op dan en slechts dan als een gegeven rij of kolom correct ingevuld is volgens de nummers. Deze methode wordt gebruikt voor het checken of een puzzel correct opgelost is in de methode *checkPuzzle()* en voor het weergeven van de groene hints langs de rand van de puzzel.

De methode *checkNumber (RowType type, int index)* levert true op dan en slechts dan als het aantal ingekleurde hokjes in een rij of kolom kleiner of gelijk is aan het aantal hokjes dat volgens de cijfers ingekleurd moet worden. Deze methode wordt gebruikt voor het weergeven van de rode hints langs de puzzel. *setColor(Color color, int x, int y)* wordt gebruikt om een driehoekje van kleur te veranderen.

reset() maakt heel het grid leeg en *setFinished()* wordt aangeroepen als de puzzel opgelost is en maakt alle lege hokjes wit. Verder worden *toString()* en *readString()* gebruikt voor het opslaan van de voortgang van een puzzel. Daarnaast zijn er weer een aantal getters die we niet hebben opgenomen in het UML-diagram zoals *getNumbers (RowType type)*, *getNumbers (RowType type, int index)*, *getA()*, *getB()*, *getC()*, *getD()*, *getE()*, *getF()*, *getStartTriangle (RowType type, int index)* en *getTriangle (int x, int y)*.

Puzzle is een *Observable* zodat als er iets verandert aan de puzzel meteen alle *Observers* geupdated worden. In ons geval zijn dat de *TriddlerView* en *SelectionView*.

View

Voor onze app hebben we twee costum views aangemaakt: *TriddlerView* en *SelectionView*. *TriddlerView* is verantwoordelijk voor het weergeven van de puzzel en *SelectionView* voor het weergeven van het kleurenselectie-menu. Beide implementeren het *Observer* interface en worden gekoppeld aan de *Observable Puzzle* zodat beide geupdated worden als er iets in *Puzzle* verandert.

TutorialView is een uitbreiding van *TriddlerView* en wordt gebruikt in de tutorial. *ZoomView* regelt het in- en uitzoomen op de puzzel. Verder hebben we nog de klasse *Style* die een aantal publieke statische attributen bevat zoals de grootte van de text, lijnbreedte en de gebruikte kleuren. Zo kunnen we deze waardes gemakkelijk aanpassen als dat nodig is.

TriddlerView
TriddlerView tekent het grid, de nummers en de groene of rode aanwijzingen langs de rand van de puzzel als een rij of kolom respectievelijk goed of fout is ingevuld. Hiervoor moeten een heel aantal waardes berekend en opgeslagen worden, zoals *puzzleWidth*, *puzzleHeight*, *triangleWidth*, *triangleHeight*, *offsetx*, *offsety*, *endGame, puzzle*, *bitmap*, *grid* en *hint*.

puzzleWidth en *puzzleHeight* zijn de breedte en hoogte van de puzzel in aantal driehoekjes, inclusief de nummers aan de rand. Met deze waardes kan, op het moment dat bekend is hoe breed en hoog de view is, de waardes voor *triangleWidth* en *triangleHeight* berekend worden. *offsetx* en *offsety* zijn de x- en y-coordinaten van het begin van het grid. Deze worden gebruikt bij het tekenen en in de *TriddlerController* voor het bepalen waar geklikt is. *endGame* houdt bij of de puzzel wel of niet opgelost is en heeft in eerste instantie de waarde *false*.

Verder wordt een verwijzing naar de puzzel opgeslagen zodat de dimensies, nummers en kleuringen van de driehoeken opgevraagd kunnen worden. *bitmap, grid* en *hint* worden gebruikt om het tekenen efficiënter te maken. Zo wordt in *bitmap* een plaatje getekend van het lege grid en alle nummers langs de rand, zodat deze later hergebruikt kan worden. *grid* bevat alle locaties en vormen van het grid van driehoekjes en *hint* bevat alle locaties en vormen van de hints langs de rand van de puzzel.

Op het moment dat *setPuzzle (Puzzle puzzle)* wordt aangeroepen wordt de verwijzing naar de puzzel opgeslagen en worden *puzzleWidth* en *puzzleHeight* uitgerekend. Aangezien de puzzel niet ondertussen verandert hoeven we dit maar één keer te doen. *onSizeChanged (int w, int h, int oldw, int oldh)* wordt vanzelf aangeroepen op het moment dat de view van grootte verandert, dit gebeurt bijvoorbeeld de eerste keer dat de view aangemaakt wordt of als het scherm wordt gedraaid. In deze methode kunnen we met behulp van *puzzleWidth* en *puzzleHeight* uitrekenen hoe breed en hoog onze driehoekjes zijn en kunnen we *offsetx* en *offsety* bepalen.

Verder tekenen we de bitmap van het lege grid en de cijfers en berekenen we alle locaties en vormen van de driehoekjes in het grid en van de hints langs de rand van de puzzel. Dit zijn alle elementen van de puzzel die weergegeven moeten worden die onveranderd blijven, tenzij de grootte van de view aangepast wordt. Door dit in *onSizeChanged* te doen in plaats van in *onDraw* hebben we de *onDraw* methode aanzienlijk sneller kunnen maken. In de *onDraw* methode kunnen we dan onze bitmap tekenen met daar overheen het tot zover ingevulde grid en de hints langs de rand.

Controller

De module controller bestaat uit een *MainController* en twee sub-controllers: *TriddlerController* en *SelectionController*, die beide een verwijzing terug bevatten naar de *MainController. MainController* bevat de puzzel als attribuut en de huidige geselecteerde kleur. Beide sub-controllers implementeren het *onTouchListener* interface.

TriddlerController berekent aan de hand van de x- en y-coordinaat van waar geklikt is welke driehoek met de huidige geselecteerde kleur gekleurd moet worden en geeft deze verandering door aan de puzzel. *SelectionController* verandert de geselecteerde kleur als hierop geklikt is.

Activities

Onze app bevat vier activities: *MainMenu*, *PuzzleSelection*, *PuzzleScreen* en *TutorialActivity*. Dit zijn dan ook de vier verschillende schermen waar de gebruiken zich kan bevinden. *MainMenu* is het beginscherm, waar gekozen kan worden om naar het puzzelselectie-menu te gaan of om naar de tutorial te gaan. *PuzzleSelection* is het puzzelselectie-menu, *PuzzleScreen* het scherm waarin een puzzel opgelost kan worden en *TutorialActivity* is de tutorial.

Vanuit het hoofdmenu *MainMenu* wordt de activity *PuzzleSelection* gestart als op de knop “Start met puzzelen” wordt geklikt of wordt de activity *TutorialActivity* gestart als op de knop “Tutorial” wordt geklikt. In *PuzzleSelection* kan een puzzel gekozen worden om opgelost te worden. Wordt hier een puzzel aangeklikt dan wordt de gekozen puzzel geladen uit een file met behulp van *PuzzleReader*, waarna een activity *PuzzleScreen* wordt gestart met deze puzzel. In *PuzzleScreen* kan deze dan opgelost worden. Wordt deze activity gesloten dan wordt de voortgang van de puzzel opgeslagen.

Ontwerpverantwoording

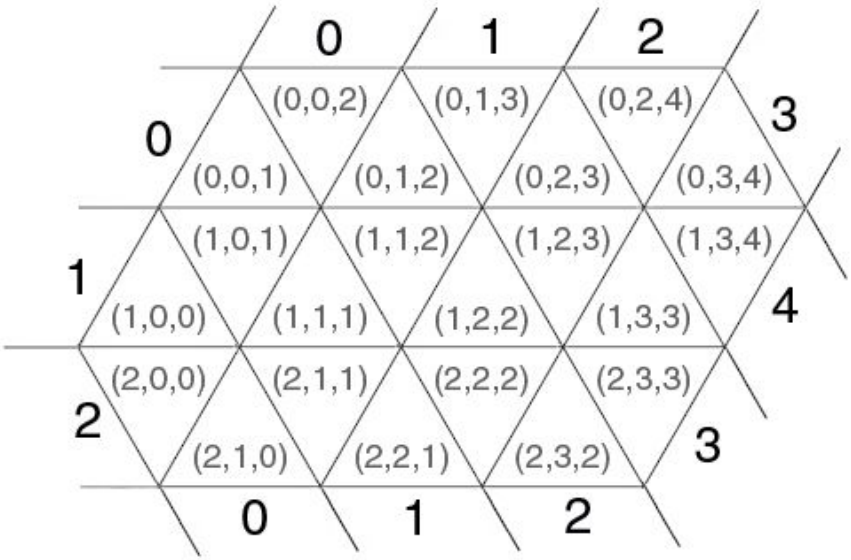
Gezien het vrijwel onmogelijk is alle keuzes die we hebben gemaakt tijdens de ontwikkeling van de app te verantwoorden, lichten we hieronder twee belangrijke keuzes die we hebben moeten maken toe. De eerste ontwerpbeslissing betreft de datastructuur van het grid (het veld met driehoekjes), de tweede betreft het opslaan van puzzels, waarbij het zowel om het inlezen van een puzzel als om de voortgang opslaan van een puzzel gaat.

Datastructuur van het grid

Het eerste probleem waar we tegenaan liepen bij het maken van onze app was hoe we het grid van een puzzel moesten representeren. Dit is namelijk niet zo triviaal als bijvoorbeeld een standaard tweedimensionaal rechthoekig grid. Hiervoor hebben we eerst een aantal overwegingen gemaakt, waarna we uiteindelijk een datastructuur gevonden hebben die werkt voor onze app.

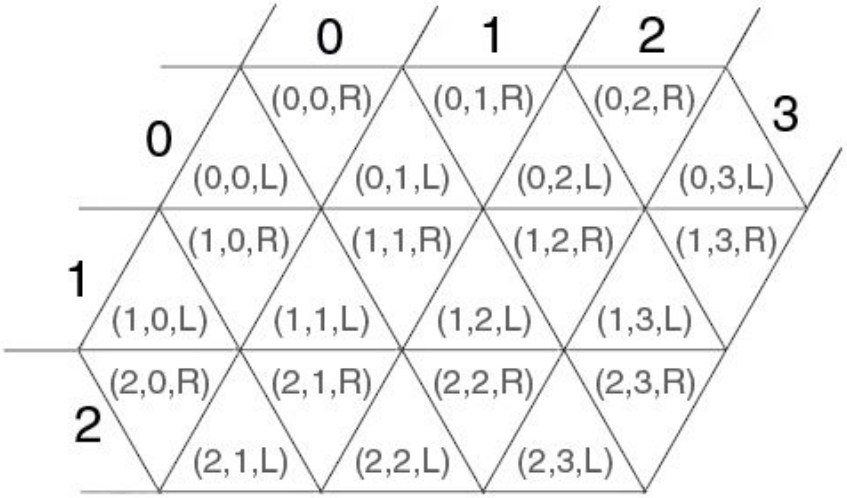
Het eerste idee was om het grid te representeren met behulp van een driedimensionale array. Immers, een tweedimensionaal rechthoekig grid is gemakkelijk te representeren in een tweedimensionale array. Nu we een grid met drie richtingen hebben lijkt het logisch om dit uit te breiden met een derde dimensie. Voor de eerste index gebruiken we dan de *row*-index, voor de tweede de *column_down*-index en voor de derde de *column_up*-index. Zo zouden we snel en gemakkelijk, gegeven indices voor *row*, *column_down* en *column_up*, het bijbehorende driehoekje kunnen bepalen.

Wat echter niet zo gemakkelijk gaat is om vanaf een driehoekje in één van de zes richtingen te gaan lopen. Starten we in het plaatje bijvoorbeeld bij (0,0,1) en lopen we herhaaldelijk naar rechts, dan zien we dat de *column_down*- én *column_up*-indices veranderen. Hetzelfde geldt als we in andere richtingen lopen. Voor het bepalen of een bepaalde rij of kolom correct is ingevuld zouden we dit wel graag gemakkelijk kunnen. Er zijn ook indices waarbij de driehoek buiten de puzzel valt. Zo behoort (0,0,0) in het plaatje hieronder bijvoorbeeld niet tot de puzzel.



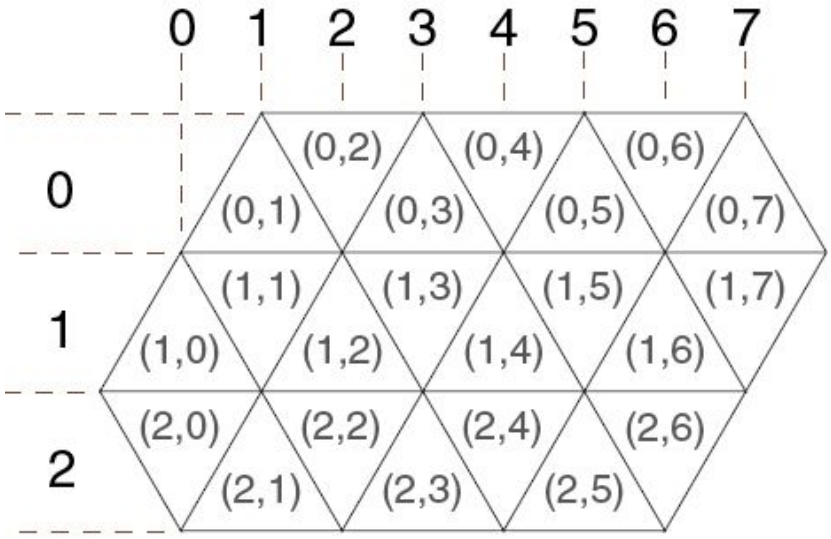
Figuur 7
Gridrepresentatie met een driedimensionale array.

Verder hebben we nog twee andere array-representaties overwogen. Eén waarbij we in plaats van de *column_up*-index bij elk driehoekje aangeven of het links of rechts in een hokje valt (ofwel of het een *up_triangle* of *down_triangle* is), zoals weergegeven in het plaatje hieronder.



Figuur 8
De tweede overwogen array-representatie. Hierbij zijn er twee indices: de eerste index loopt over alle rijen en de tweede over alle kolommen die van rechtsboven naar linksonder lopen. *L* geeft aan dat een driehoek links in een hokje valt en *R* rechts.

In de derde representatie gebruiken we twee indices, de eerste voor de rij waarin een driehoekje zich bevindt en de tweede voor de verticale kolom waarin een driehoekje zich bevindt.



Figuur 9
De derde overwogen array-representatie met twee richtingen.

Eigenlijk hebben alle drie van deze array-representaties hetzelfde probleem: het is lastig om te bepalen waar een rij of kolom begint, het is lastig om over een bepaalde rij of kolom te lopen en het is lastig om te bepalen of, gegeven twee of drie indices, deze driehoek wel of niet bestaat in het grid. Wij vonden dat dit makkelijker moest kunnen. Uiteindelijk zijn we op de datastructuur gekomen zoals beschreven in het model, waarbij elke driehoek gelinkt is aan zijn buren.

In *Puzzle* hebben we dan alle driehoeken in het grid correct aan elkaar gekoppeld en slaan we alle driehoeken aan de rand op. Zo kunnen we én gemakkelijk over een rij of kolom lopen én gemakkelijk de eerste driehoek van een rij of kolom opvragen. Zo hoeven we ons ook niet druk te maken over waar een rij of kolom begint.

Het enige wat iets lastiger is geworden is om een bepaalde driehoek in het grid op te vragen of om deze te kleuren. Hiervoor hebben we uiteindelijk tóch de derde array-representatie gebruikt. De *TriddlerController* bepaalt aan de hand van waar is geklikt welke indices in de derde representatie hierbij horen. In de methode *setColor (Color color, int x, int y)* in Puzzle geven we dan het juiste driehoekje terug. Uiteindelijk was dit vele malen simpeler dan als we alleen een van de array-representaties hadden gebruikt.

Opslaan van puzzels

In ons plan voor de app hadden we ook het opslaan van puzzels opgenomen. Dit bleek echter een grote uitdaging te zijn.

Ons eerste idee was om het puzzelobject in zijn geheel op te slaan in een bestand. Dit wilden we voor elke puzzel eenmalig doen en dit bestand telkens bijwerken wanneer de puzzel werd aangepast. Dit probeerden we door een klasse aan te maken die het interface *Serializable* implementeerde. Dit bleek echter complex, omdat een object - en daarmee dus de gehele datastructuur - opslaan geen standaard feature is in android. Daarbij is het ook niet gemakkelijk te implementeren. We hebben dit geprobeerd, maar hebben uiteindelijk gekozen om gebruik te maken van de sharedPreferences.

Wat we hebben gedaan is het volgende: elke puzzel wordt in de app opgeslagen in een textbestand. Hierin worden de lengtes van de rijen aangegeven en welke nummers er naast de rijen moeten komen te staan. Wanneer een puzzel deels is opgelost, wordt deze puzzel opgeslagen als String in de sharedPreferences. Bij volgend gebruik van de puzzel wordt deze weer geladen vanuit de preferences.

We hebben hiervoor gekozen omdat het makkelijker te implementeren was, maar ook omdat het sneller was. Het laden van een puzzelobject uit een bestand duurde aanzienlijk langer dan via de sharedPreferences. Wat we niet hebben getest is of het gebruik hiervan problemen zou opleveren bij een database met meer puzzels.

Het gebruik van de sharedPreferences had nog een ander bijkomend voordeel. Het was hierdoor makkelijker om in het selectiescherm van de puzzels aan te geven of een puzzel is opgelost. Als we dit hadden willen doen met het opslaan van het gehele object had dit wederom aanzienlijk veel tijd gekost, omdat alle puzzels ingelezen zouden moeten worden bij het maken van het overzicht. Met de sharedPreferences kost dit niet merkbaar veel tijd.

Reflectie

Toen we besloten dat we een app voor triddlers gingen schrijven, waren we erg ambitieus. We hadden bedacht dat een gebruiker met onze app in staat zou moeten zijn om:

- een puzzel op te lossen
- tijdens het oplossen in een puzzel te kunnen inzoomen
- zijn voortgang te kunnen opslaan
- een nieuwe puzzel aan te kunnen maken
- te bepalen of een zelfgemaakte puzzel uniek oplosbaar is
- een zelfgemaakte puzzel te kunnen delen met anderen

Nadat we dit idee aan de begeleiders en andere projectgroepen gepresenteerd hadden, werd ons verteld dat dit waarschijnlijk een lastig project zou worden. Toen hebben we overwogen om dan een app voor Japanse puzzels te maken, hoewel dit veel minder interessant zou zijn, omdat er al veel apps voor Japanse puzzels bestaan. Uiteindelijk zijn we met dit probleem naar onze docent Sjaak Smetsers gegaan. Hij heeft ons verteld dat het niet erg zou zijn als we een app zouden maken die minder features heeft dan we oorspronkelijk hadden bedacht, maar dacht dat het maken van een app voor triddlers meer zou toevoegen dan een app voor Japanse puzzels.

We hebben zodoende besloten toch een app voor triddlers te schrijven, maar dan zonder bepaalde features. Zo hebben we van de volgende features afgezien: het zelf aanmaken van een puzzel, controleren of een zelf gemaakte puzzel uniek oplosbaar is en een zelfgemaakte puzzel naar anderen kunnen sturen.

Naderhand zijn we blij dat we deze keuze hebben gemaakt: het was een erg leuk project om aan te werken en we zijn erg tevreden met ons resultaat. Onze app functioneert goed en ziet er ook mooi uit. Daarnaast is de code leesbaar en goed geordend, hoewel een gebruiker van de app dit niet zal zien.

De samenwerking binnen de groep verliep erg goed. We hebben van tevoren een vast aantal uren afgesproken waarop we elke week bij elkaar kwamen om aan de app te werken. Over het algemeen hebben we het meeste werk in (wisselende) tweetallen gedaan en dit werkte prima.

Tijdens het maken van de app hebben we veel geleerd, zoals het gebruik maken van het MVC-model en hoe je bestanden (deels ingevulde puzzels in ons geval) op kan slaan op de telefoon van de gebruiker van de app. Dit laatste bleek lastiger dan verwacht. Ook het implementeren van de zoom-functie bleek erg lastig te zijn, maar dit is uiteindelijk wel gelukt.

Een nadeel aan onze app is dat er op dit moment maar 24 puzzels in zitten (en alleen het tekenen en maken van die 24 puzzels kostte al veel tijd). Hierdoor kan een gebruiker na redelijk korte tijd niet zo veel meer met onze app, omdat hij alle puzzels al heeft opgelost. Het zou erg leuk zijn als er een optie zou zijn om zelf een puzzel te maken en deze vervolgens te kunnen delen met anderen, maar het was met de gegeven tijd gewoon niet mogelijk om dit te implementeren.

Wat wij zelf erg fijn vonden was dat we erg veel vrijheid hadden om zelf te experimenteren met het schrijven van onze app. Bij onze groep werkte dit erg goed. Al met al was dit project een leuke eerste ervaring voor het maken van een app. Als we later nog eens een app zouden moeten schrijven, zouden we waarschijnlijk nog steeds een aantal dingen moeten opzoeken op internet, maar we weten nu hoe we moeten beginnen en we hebben natuurlijk veel geleerd over de basis van het schrijven van een app.