osu!mania voor Android

26-6-2015

Oussama Danba, Erin van der Veen en Kris Elsinga

Inhoudsopgave

Voorwoord

Dit document beschrijft de osu!mania app die we hebben gemaakt. Het dient als een overzicht op de ontwikkeling van de osu!mania app en laat zien welke problemen wij zijn tegengekomen. Daarnaast willen we enkele ontwerpbeslissingen ter sprake brengen en beargumenteren.

Het document zal beginnen met een introductie van de osu!mania app en wordt gevolgd door de productverantwoording en specificaties. Daarna komt het ontwerp van de app en de ontwerpbeslissingen die zijn genomen. We sluiten af met de reflectie en de appendix.

Beschrijving

Inleiding

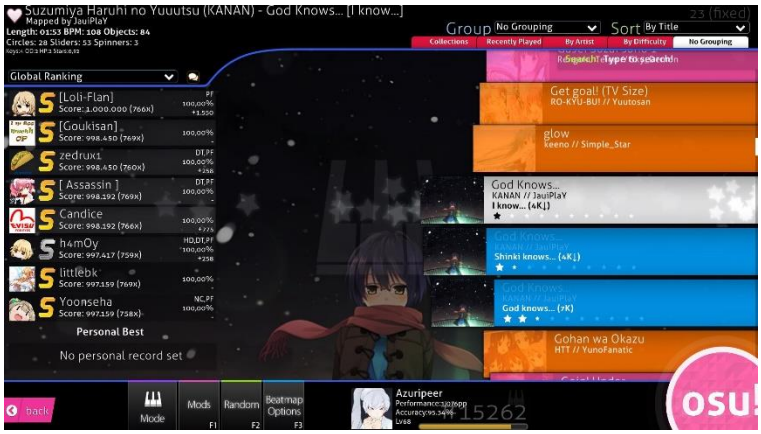
Osu!mania is gemaakt als Android-variant van het gelijknamige computerspel. Er is dan ook geprobeerd om de belangrijkste eigenschappen van de computervariant te behouden. Bovendien is er aandacht besteed aan het behouden van het thema.

Als men het oorspronkelijke computerspel opstart krijgt men het volgende te zien.

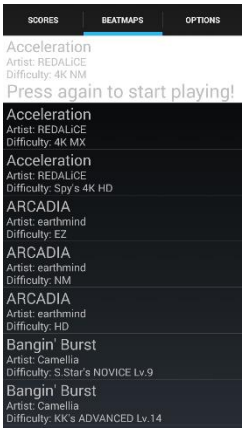
Een simpele achtergrond met in het midden het osu! logo dat ritmisch op de muziek beweegt. Er is op dit punt maar één interactie vanzelfsprekend en dat is op het logo drukken.

Wanneer dit wordt gedaan verschijnt er een viertal knoppen. Er is geprobeerd om dit over te nemen in de app, maar wegens tijdgebrek is er veel van het originele menu geabstraheerd.

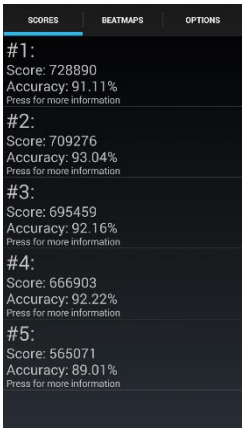
Het huidige menu is hierboven te zien. Wanneer men op het logo drukt verschijnt het viertal knoppen van de computervariant niet. Men komt meteen in het *beatmap selection menu* waar men een *beatmap* kan kiezen. In de computervariant kan dit bereikt worden door op “Play” te drukken.



In het *beatmap selection menu* wordt, mits de standaardinstellingen zijn gekozen, een lijst gesorteerd op titel van het nummer. Wanneer men op een nummer drukt wordt dit nummer geselecteerd en krijgt men de verschillende moeilijkheidsgradaties van het nummer te zien. Bovendien wordt links in beeld een lijst met topscores getoond. Vanwege het relatief kleine scherm in de meeste Android toestellen is ervoor gekozen om dit in de app iets anders aan te pakken.

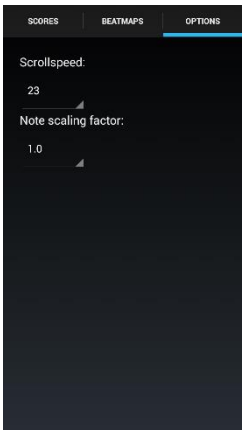


Acceleration is wit, dit betekend dat deze *beatmap* op het moment geselecteerd is. Wanneer men een *beatmap* heeft geselecteerd is het mogelijk om naar “Scores” swipen om de scores van de geselecteerde *beatmap* bekijken.



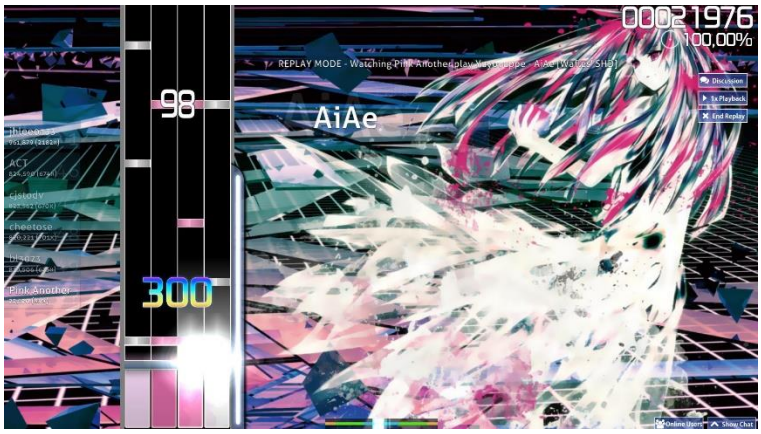
De scores worden gesorteerd en in aflopende volgorde getoond, ditzelfde gebeurt ook in de computervariant. Daarnaast is de accuracy (nauwkeurigheid) beschikbaar en kan men op een score drukken om details van deze score te bekijken.

Wanneer er op “Options” wordt gedrukt verschijnt het volgende scherm:

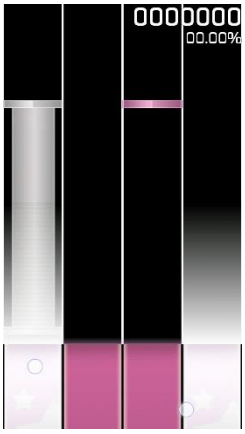


De computervariant bevat velen malen meer opties, maar wegens tijdgebrek is ervoor gekozen om alleen de belangrijkste opties over te nemen. *Scrollspeed* bepaalt hoe snel de noten naar beneden komen, ieder heeft hier zijn eigen voorkeur in. Op een tablet of ander groot apparaat kan het verstandig zijn om deze hoger in te stellen in vergelijking met een kleiner apparaat. *Note scaling factor* is aan de app toegevoegd om het spel ook goed speelbaar te maken op apparaten met een hoge dpi. Deze optie bepaald hoe groot de noten worden weergegeven. Dit maakt het spel echter niet makkelijker, het blijft even moeilijk om de noten correct in te drukken.

Na het kiezen van de *beatmap* komt men zowel in de computervariant als in de app in het spel terecht. Aan dit deel van de app is het meeste tijd besteed, dit is immers het belangrijkste deel. De computervariant ziet er zo uit:



In de app is gekozen voor een minimalistisch uiterlijk vanwege het relatief kleine scherm waarmee gewerkt moet worden.



De belangrijkste elementen zijn overgenomen uit het computerspel. De score bovenin, het licht van de toetsen, de accuracy als percentage, de score die je krijgt als je een noot raakt en het opslaan van de score in het scoremenu.

Productverantwoording

Het concept achter de computervariant van osu!mania is niet helemaal nieuw. De computervariant van osu!mania is gebaseerd op het arcadespel *beatmania*. Een afgeleide hiervan is het spel *Dance Dance Revolution*. Er zijn op het moment al verschillende Android-varianten aanwezig welke gebaseerd zijn op *Dance Dance Revolution*. Er zijn echter geen Android-varianten van *beatmania* en osu!mania; de Android-varianten van *Dance Dance Revolution* zijn vaak wel in staat om *beatmaps* te kunnen converteren naar hun eigen variant. Onze variant van osu!mania onderscheidt zich van deze varianten door compatibel te zijn met *beatmaps* die gemaakt zijn voor de computervariant van osu!mania. Dit heeft als gevolg dat het niet nodig is om aparte *beatmaps* te maken voor de Android-variant van osu!mania en dat spelers al bekend zijn met de *beatmaps* omdat ze niet geconverteerd worden.

Specificaties

Functionele eigenschappen

De volgende functionele eigenschappen moest de app volgens ons hebben:

ID	Beschrijving
FE1	Gebruiker kan beatmap selecteren
FE2	Gebruiker kan beatmap starten
FE3	Gebruiker kan spel spelen door op de toetsen te drukken
FE4	Gebruiker kan spel pauzeren
FE5	Gebruiker kan uit het spel gaan en terugkomen zonder dat zijn voortgang is verdwenen
FE6	Gebruiker kan zijn scores bekijken
FE7	Gebruiker kan gedetailleerde informatie over een score opvragen
FE8	Gebruiker kan het spel aanpassen naar zijn eigen wensen
FE9	Gebruiker kan nieuwe beatmaps toevoegen

Vanwege het hoge aantal functionele eigenschappen zullen wij ons limiteren tot vijf use-cases.

ID	UC01
Naam	Start spel
Functionele eigenschap	FE2
Uitleg	Gebruiker kiest een beatmap en start het spel op
Precondities	De app is opgestart en bevindt zich in het hoofdmenu.
Resultaat	Het spel is opgestart.
Loop van gebeurtenissen	1. Gebruiker drukt op het osu!mania logo en wacht tot de app klaar is met laden. 2. Gebruiker gaat door de lijst heen en kiest een beatmap. 3. Gebruiker drukt weer op de beatmap om het spel te starten.

ID	UC02
Naam	Multitasking
Functionele eigenschap	FE5
Uitleg	Tijdens het spelen vertrekt de gebruiker uit het spel. Later komt hij terug en wil hij verdergaan.
Precondities	Een beatmap is gekozen en een gebruiker bevindt zich aan het begin van het spel.
Resultaat	De gebruiker gaat verder waar hij is gebleven.
Loop van gebeurtenissen	1. Gebruiker speelt de beatmap tot een willekeurig punt. 2. Gebruiker drukt op de home knop om uit de app te gaan. 3. Gebruiker komt later terug in de app. 4. Gebruiker ziet een pauzescherm en drukt op “Continue” om verder te gaan.

ID	UC03
Naam	Score bekijken
Functionele eigenschap	FE7
Uitleg	De gebruiker heeft net een score gehaald en wil weten hoeveel keer hij een regenboog driehonderd heeft gehaald.
Precondities	Een beatmap is geselecteerd en er is een score geplaatst bij deze beatmap.
Resultaat	De gebruiker kan zien hoeveel keer hij een regenboog driehonderd heeft gehaald.
Loop van gebeurtenissen	1. Gebruiker swiped naar Scores toe of drukt op het woord Scores linksboven. 2. Gebruiker gaat door de lijst heen. 3. Gebruiker drukt een score aan om meer informatie te krijgen.

ID	UC04
Naam	Scrollspeed aanpassen
Functionele eigenschap	FE8
Uitleg	De noten komen te langzaam naar beneden en de gebruiker wil dit versnellen.
Precondities	De gebruiker bevindt zich in het beatmap selection menu.
Resultaat	Scrollspeed is verhoogd.
Loop van gebeurtenissen	1. Gebruiker swiped naar Options toe of drukt op het woord Options rechtsboven in het scherm. 2. Gebruiker drukt de huidige scrollspeed aan. 3. Gebruiker gaat door de lijst heen en kiest een hogere scrollspeed.

ID	UC05
Naam	Beatmaps toevoegen
Functionele eigenschap	FE9
Uitleg	De gebruiker heeft een beatmap gedownload en wil deze spelen.
Precondities	De beatmap die de gebruiker wil spelen is al gedownload.
Resultaat	De gebruiker kan de beatmap zien in het beatmap selection menu.
Loop van gebeurtenissen	4. Gebruiker navigeert naar het .osz bestand. 5. Gebruiker drukt op het bestand waardoor de app opent. 6. Gebruiker wacht totdat de app klaar is met opstarten en gaat naar het beatmap selection menu om zijn beatmap te zien.

Niet-functionele eigenschappen

De voornaamste niet functionele eigenschap van de app is uiteraard dat het spel leuk is om te spelen. Uit deze eigenschap volgt automatisch de overige niet-functionele eigenschappen. Een spel is bijvoorbeeld alleen leuk om te spelen wanneer dit spel vloeiend loopt, een gebruiker zal zich snel ergeren wanneer het spel hapert, of in extreme gevallen zelfs vastloopt. Bovendien is dit voor osu!mania nog meer van belang als bij menig andere app. Wanneer het spel hapert, zal de gebruiker een

noot kunnen missen terwijl er wel op het juiste moment op het scherm gedrukt werd. Gebruiksgemak is uiteraard ook een belangrijke eigenschap. Hier is echter minder aandacht aan besteed omdat de prioriteit bij het maken van een functionele app lag.

Behalve deze belangrijke niet-functionele eigenschappen, is er uiteraard ook een aantal eigenschappen waar minder de nadruk op werd gelegd tijdens het ontwikkelen van de app. De app moest bijvoorbeeld stabiel werken, al is dit wel een gevolg van de eerdere eigenschap dat het spel vloeiend moet lopen.

Naast de hiervoor genoemde eigenschappen heeft onze app ook eigenschappen die inherent zijn aan het bouwen op een bestaande community. Zo is de app heel gemakkelijk uit te breiden en is de documentatie van de app erg uitgebreid.

Ontwerp

Globaal ontwerp

De osu!mania app heeft drie hoofdcomponenten welke uit kleinere componenten bestaan. Het eerste hoofdcomponent en het simpelste hoofdcomponent is het startscherm. Zodra de app wordt opgestart komt de gebruiker hierin terecht en is er maar één optie; op het osu!mania logo drukken. Aanvankelijk was het de bedoeling dat er een optiemenu zou komen maar vanwege tijdgebrek is er besloten om dit toch niet te doen. Als er nu op het logo wordt gedrukt wordt hoofdcomponent twee geladen. Er zijn wel enkele andere redenen om het component nog te laten bestaan. Ten eerste, als hoofdcomponent twee wordt gestart moet er even gewacht worden totdat de app klaar is met laden. Zonder hoofdcomponent één begint de app meteen met laden en kan de gebruiker de indruk krijgen dat de app is vastgelopen. Ten tweede wordt het component ook gebruikt voor het uitpakken van *beatmaps*. Als de gebruiker een bestand met de extensie “osz” opent start de app op en wordt de *beatmap* uitgepakt naar de osu!mania map en zal de app het originele bestand verwijderen. Ten slotte is het erin gelaten omdat het niet zeker was of er nog aan gewerkt ging worden.

Hoofdcomponent twee bestaat uit drie kleinere componenten waartussen geswiped kan worden. Deze drie componenten zijn “Scores”, “Beatmaps” en “Options”. De gebruiker komt als eerste terecht in het *beatmaps* component. Deze component toont een lijst met beschikbare *beatmaps* waaruit de gebruiker kan kiezen. Om een *beatmap* te spelen moet de gebruiker eerst de gewenste *beatmap* selecteren en dan nog eens aandrukken. Het resultaat is dat de gebruiker naar hoofdcomponent drie gaat. Nadat een gebruiker een selectie heeft gemaakt kan de gebruiker er ook voor kiezen om naar links te swipen waardoor het scores component getoond wordt. Hier krijgt de gebruiker een lijst te zien van eerder behaalde scores op de geselecteerde beatmap. Een gebruiker kan op een score drukken om gedetailleerde informatie te zien. De gebruiker kan er ook voor kiezen om naar het options component te gaan. Hier is het mogelijk om enkele opties aan te passen zodat de gebruiker de app beter kan ervaren. Bovenaan is altijd een menu te zien waaruit blijkt bij welk component de gebruiker op het moment is en de gebruiker kan dit menu gebruiken om naar een component te gaan zonder te swipen.

Hoofdcomponent drie is het daadwerkelijke spel. De gebruiker kan hier een beperkt aantal acties uitvoeren. De gebruiker kan in een van de vier kolommen drukken, de gebruiker kan op de terug knop drukken en de gebruiker kan uit de app gaan door middel van de home knop in te drukken of het apparaat te vergrendelen. Een toets indrukken gebeurt door in een kolom te drukken en is nodig voor de interactie met het spel. Het indrukken van de terug knop resulteert in een pauzescherm waar drie

opties mogelijk zijn. De eerste optie is het hervatten van het spel, de tweede optie is het herstarten van het spel en de derde optie is het afsluiten van het spel wat ervoor zorgt dat de gebruiker terug gaat naar hoofdcomponent twee. Als de speler afgaat in het spel gaat hij ook terug naar hoofdcomponent twee. Het indrukken van de home knop en het vergrendelen van het apparaten zorgt er ook voor dat het pauzescherm verschijnt.

Detailontwerp

De app maakt gebruik van het MVC design pattern. De model, view en controller zijn gescheiden en bevinden zich in aparte packages. De app bevat in totaal 24 klassen waarvan drie bij hoofdcomponent één horen, tien bij hoofdcomponent twee horen en elf bij hoofdcomponent drie horen. Hoofdcomponent één en twee zijn niet heel bijzonder en zullen we hier dus ook niet beschrijven. Hoofdcomponent drie daarentegen is erg complex en zullen we hier wel uitvoerig bespreken.

De activity die bij hoofdcomponent drie hoort is de GameActivity. Deze klasse breidt Activity natuurlijk uit en implementeert Observer. De reden dat GameActivity Observer implementeert is zodat de GameActivity weet wanneer de game is afgelopen. Deze klasse bevat de methoden onResume, onPause, onBackPressed en onCreate. De eerste drie hiervan worden gebruikt voor het pauzemenu en onCreate wordt gebruikt voor het opzetten van de GameActivity. Tijdens het opzetten worden de SharedPreferences opgehaald zodat het spel weet welke opties de gebruiker heeft gekozen. Daarnaast wordt de GameState aangemaakt en wordt de GameActivity als Observer toegevoegd aan de GameState. Ook worden de GameStateController en de GameView aangemaakt. Ten slotte wordt de GameState in een thread opgestart.

De GameState is het daadwerkelijke spel en behoort tot het model. Deze klasse bevat 41 methoden, breidt Observable uit en implementeert Runnable. De belangrijkste methoden zijn loadInitialPositions, run en step. De rest van de methoden worden aangeroepen vanuit deze methoden en andere klassen in het geval van getters en setters. De loadInitialPositions methode wordt aan het begin aangeroepen om te bepalen wat de startposities zijn van alle noten. De run methode start de thread op en stopt hem zodra het spel is afgelopen. De step methode wordt voortdurend uitgevoerd en berekend hoe ver de noten omlaag moeten op basis van de verstreken tijd sinds de vorige keer dat de step methode werd uitgevoerd. De step methode voert daarnaast nog enkele andere taken uit zoals het uitvoeren van TimingPoints, gemiste noten als mis tellen, animatiestadia opnieuw uitrekenen en de accuracy opnieuw uitrekenen.

GameStateController is de controller van de GameView en heeft één methode genaamd onTouch. Deze methode handelt de input van de gebruiker af en kan omgaan met meerdere vingers. Zodra een gebruiker op het scherm drukt of zijn vinger loslaat wordt de GameState hiervan op de hoogte gesteld zodat deze kan bepalen of hij op het juiste moment heeft gedrukt of losgelaten.

De GameView is in de app geen ordinaire view maar een SurfaceView die in een thread zit. De reden hiervoor komt later in de ontwerpverantwoording. GameView heeft twintig methoden die het gebruikt om de bitmaps te initialiseren en te tekenen op het scherm. In de constructor worden alle bitmaps geladen en eventueel geschaald. Daarnaast worden de toetsen gecreëerd en de GameViewThread wordt opgestart. De belangrijkste methode is de doDraw methode, deze tekent namelijk alle onderdelen op het scherm elke keer als het scherm wordt ververst. Als het spel gepauzeerd is wordt het pauzemenu eroverheen getekend.

Ontwerpverantwoording

Gebruik SurfaceView in thread

Twee belangrijke doelen waren het voorkomen van haperingen en het spel moet vloeiend spelen. Deze eisen lijken heel erg op elkaar maar verschillen subtiel. Met haperingen bedoelen we visuele haperingen en met vloeiend spelen bedoelen we de tijd tussen het fysiek drukken en de registratie ervan in het spel. Aan visuele haperingen kunnen wij niet heel veel doen aangezien het simpelweg kan zijn dat een apparaat niet krachtig genoeg is, maar wij kunnen het spel wel vloeiender laten spelen. De doDraw methode in de GameView wordt elke keer uitgevoerd als het scherm ververst. Dit is op de meeste Android toestellen 60 Hertz, dat betekent dus dat het scherm ongeveer elke 17 milliseconden ververst. Omdat de controller verbonden is aan de GameView betekent dit dat de controller ook eens in de 17 milliseconden input registreert. Dit was voor ons helaas niet voldoende omdat de regenboog driehonderd score een periode heeft van 16 milliseconden voor en na de noot. Er is dus een tijdsbestek van 32 milliseconden in totaal, wat als resultaat heeft dat het voor de gebruiker bijna onmogelijk is om een regenboog driehonderd te kunnen halen.

Dit hebben wij opgelost door een SurfaceView te gebruiken in een aparte thread. De SurfaceView heeft als eigenschap dat het kan tekenen zonder dat het op de UI thread zit. Door de SurfaceView in een aparte thread te zetten wordt de doDraw methode vaker uitgevoerd. Er wordt weliswaar maar eens in de 17 milliseconden op het scherm getekend omdat dit een fysieke limitatie is maar de controller verwerkt nu wel vaker input van de gebruiker. Hierdoor hebben wij de tijd van een input kunnen reduceren tot ongeveer twee milliseconden wat het spel een stuk beter speelbaar maakt.

Omgaan met onbekende informatie

In de app wordt heel veel gebruik gemaakt van constanten die uit de lucht komen vallen en formules die niet helemaal evident zijn. Dit heeft te maken met het feit dat osu!mania nauwelijks gedocumenteerd is. Wij hebben deze formules dus zelf moeten maken door te kijken naar wat er precies gebeurt. Formules zoals basisgrootte van de noten hebben we kunnen achterhalen door het spel op verschillende resoluties uit te proberen en deze resultaten te plotten. Met behulp van deze plot kunnen wij een formule maken.

Het meest interessante gedeelte was het achterhalen van de formules omtrent levens. Als een speler een noot mist of een vijftig haalt dan verliest hij levens. Als de speler een tweehonderd, driehonderd of regenboog driehonderd haalt dan krijgt hij levens terug. Een honderd halen heeft geen effect op het aantal levens van de speler. De indicatie van het aantal levens is een meter die opgevuld is met het percentage levens dat over is. Hier kwamen wij het grootste probleem tegen, er is geen mogelijkheid om te zien hoeveel levens een speler precies heeft. Uit een meter is niet af te leiden of een gebruiker standaard honderd of meer levens heeft. Wij gingen ervanuit dat een speler begint met honderd levens. Om uit te vinden hoeveel levens een speler verliest per mis startten we het spel op en keken we hoelang het duurt voordat een speler afgaat. Door dit te doen met verschillende “HP Drain” waardes konden wij een formule maken die afhankelijk is van de “HP Drain” waarde van een *beatmap*. Hoeveel levens een speler per noot verliest kwam uiteindelijk uit op $0.75 * (drain + 1)$. Hoeveel levens een speler terugkrijgt was op deze manier niet te doen. We konden namelijk niet precies weten hoeveel levens een speler erbij heeft gekregen per noot. De oplossing die we uiteindelijk hebben gebruikt is door gebruik te maken van een programma genaamd “Cheat Engine”. Dit programma stelde ons in staat om waardes in het geheugen van het spel te bekijken en helpt ons ook om deze waardes in het geheugen te vinden.

Om het adres te vinden stelden we het programma zo in dat we niet wisten met welke waarde we begonnen; dat een speler honderd levens heeft was immers een aanname. Vervolgens mistten we opzettelijk en gaven we in het programma aan dat we op zoek gingen naar adressen waarvan de waarde was gedaald. Dit konden we voortdurend herhalen totdat er een klein aantal adressen overbleef. Daarna raakten wij expres een noot en gaven we aan dat de waarde was gestegen. Het resultaat was dat we het geheugenadres van de levens hadden gevonden. Hierna begonnen we een nieuw spel en zagen we dat het aantal levens op tweehonderd begon. Onze aanname bleek dus fout te zijn, dit betekende dus dat onze eerder opgestelde formule fout was. De correcte formule was eigenlijk $1.5 * (drain + 1)$. Nadat we het adres van het aantal levens hadden gevonden konden we gemakkelijk de rest van de formules bepalen. We gingen simpelweg alle combinaties van “HP Drain” en mogelijke score langs. Deze data hebben we verzameld en geplot om vervolgens formules van te kunnen maken, deze formules zijn uiteindelijk in onze app verwerkt. De uiteindelijke tabellen en grafieken zijn te vinden als appendix.

Reflectie

Over het spel zelf zijn we aardig tevreden, er zijn echter een paar dingen die we liever anders hadden gezien. Zo hadden we het laden van de *beatmaps* het liefst in een aparte thread gedaan en hadden we onze eigen muziekspeler willen schrijven om de vertraging die Android in de standaardspeler heeft weg te werken. Bovendien zouden we liever hebben gezien dat het spel pas start als de GameActivity ook echt klaar is met opstarten en dat het spel aftelt voordat de muziek begint met spelen.

Buiten het spel, in het menu, hadden wij vrijwel alles liever anders gedaan. De meest belangrijkste punten zijn: de stijl in het menu komt niet overeen met het computerspel, er zijn geen opties voor skins, het is niet duidelijk genoeg dat je op het logo moet drukken in het hoofdmenu, de scores worden niet mooi weergegeven, de *beatmaps* spelen hun muziek niet af wanneer ze geselecteerd zijn en de achtergrond veranderd niet naar de achtergrond van de geselecteerde *beatmap*. Zoals al eerder geschreven in dit document is dit te wijten aan tijdgebrek, veel liever hadden we nog een paar weken extra gehad om de app precies zo te maken als we voor ogen hadden.

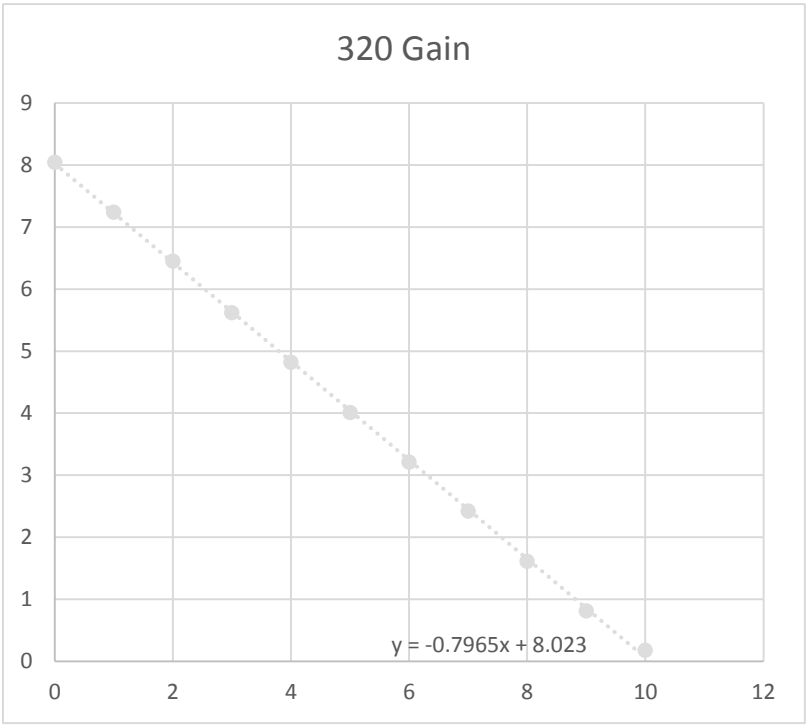
Tijdens het programmeren zijn we tegen verschillende problemen aangelopen, de grootste hiervan is waarschijnlijk de muziekspeler van Android. Deze muziekspeler speelt de muziek een heel klein beetje langzamer af op sommige apparaten. Wanneer men normaal gesproken muziek wil afspelen is dit geen probleem, maar wanneer men op de maat van de muziek iets moet doen is dit haast onmogelijk. We hebben dit moeten oplossen door elke zeventig milliseconden (dit veranderd per apparaat) uit te zoeken hoe ver de noten voor- of achterliepen en de snelheid daarop aan te passen. Zoals hierboven werd opgemerkt zouden we als we weer een Android app zouden moeten schrijven, zelf een muziekspeler maken. Aangezien dit erg lastig is, zullen we in de praktijk waarschijnlijk dus simpelweg een ander platform kiezen; Windows Phone of iOS bijvoorbeeld.

Een ander probleem waar we tegenaan liepen was de werking van het spel. De documentatie over hoe het computerspel precies werkt is onvolledig. We hebben meerdere malen formules moeten bedenken of afleiden uit testresultaten. Deze documentatie hadden we misschien ook kunnen vragen aan de oorspronkelijke maker van het spel.

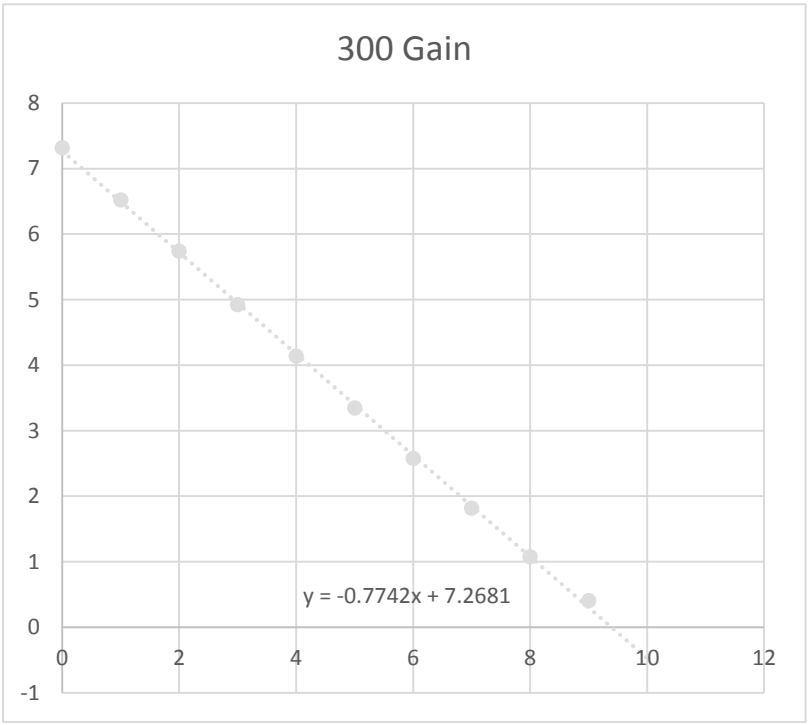
Samengevat zouden we, als we de app nog eens zouden moeten maken, meer tijd nemen, een ander platform kiezen en peppy (de maker van het originele spel) vragen of we de broncode of anders de formules van het spel mogen hebben.

Appendix

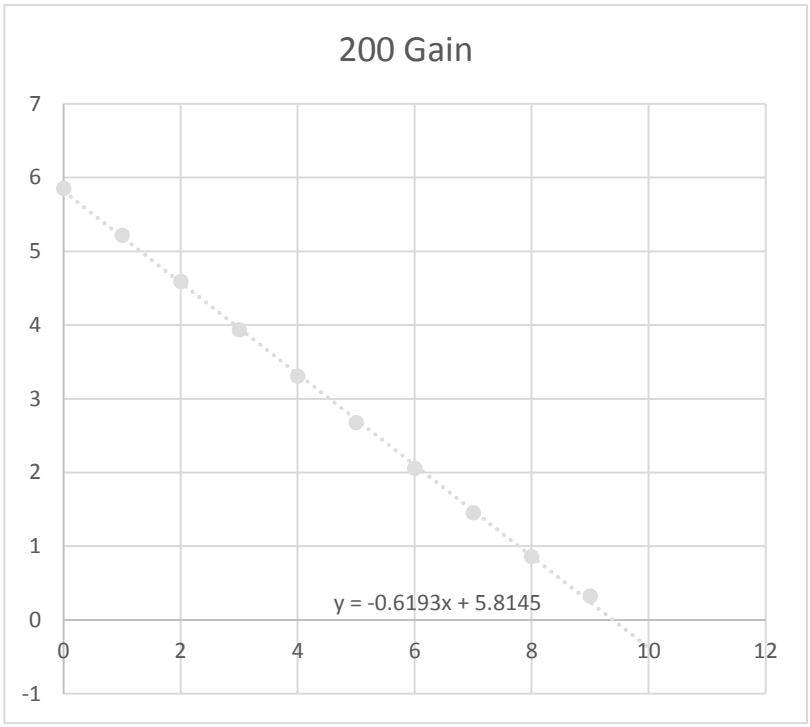
HP Drain	320 Gain
0	8.04762
1	7.243582
2	6.454677
3	5.624439
4	4.824414
5	4.013596
6	3.214156
7	2.422304
8	1.612254
9	0.813474
10	0.17809



HP Drain	300 Gain
0	7.316018
1	6.519224
2	5.737491
3	4.921385
4	4.135212
5	3.344663
6	2.571325
7	1.816728
8	1.074836
9	0.406737
10	#N/A



HP Drain	200 Gain
0	5.852814
1	5.215379
2	4.589993
3	3.937108
4	3.308169
5	2.675731
6	2.05706
7	1.453382
8	0.859869
9	0.32539
10	#N/A



HP Drain	50 "Gain"
0	-0.32
1	-0.64
2	-0.96
3	-1.28
4	-1.6
5	-1.92
6	-2.24
7	-2.56
8	-2.88
9	-3.2
10	-3.52