Cover page
Een flexibele tool voor het vertellen van diverse digitale interactieve verhalen.

Een flexibele tool voor het vertellen van diverse digitale interactieve verhalen

Auteur: Swen Meeuwes; 0887127@hr.nl
Bedrijf: &ranj

Bedrijfsbegeleider: I. Swartjes; ivo@ranj.nl
Examinator 1: I.S. Paraschiv; i.s.paraschiv@hr.nl
Examinator 2: J. Grobben; j.grobben@hr.nl

CMI - Informatica
Hogeschool Rotterdam
Nederland
28 mei 2018

Versie: 0.1

Samenvatting

Nog niet in concept versie.

Management Summary

Not available in concept version.

Inhoudsopgave

Hoofdstuk 1

Introductie

1.1 Achtergrond en aanleiding

1.2 Probleemstelling

1.2.1 De technology stack

1.2.2 Diversiteit in game content

1.2.3 Het ondersteunen van meerdere formalismen

1.2.4 Overkoepelende projectstructuur

1.3 Doelstelling

1.4 Onderzoeksvragen

1.5 Afbakening

1.6 Structuur

Hoofdstuk 2

Methodes

Om meer inzicht te krijgen in de theorie achter de narratieve bewerkers worden er interviews met experts op het gebied van narratieve serieus games afgenomen. De geïnterviewde zijn medewerkers van &ranj met de titel "senior". Verder hebben ze minimaal 3 jaar nauw met de narratieve bewerkers gewerkt waarin ze minimaal 5 serious game projecten afgerond hebben waar in een narratief het leidende mechaniek was. Door deze ervaring hebben zij een goed beeld van de huidige bewerkers en de achterliggende theorie. Dit maakt hun experts op het gebied van *narrative serious games* en dus de geschikte kandidaten voor een interview betreft het onderwerp 'narratieve bewerkers'.

Tijdens het afnemen van deze interviews worden de resultaten meegenomen en besproken bij het volgende interview. Op deze manier kan er een discussie ontstaan die niet alleen evalueerd maar ook validerend kan werken.

Hoofdstuk 3

Technologieën

In dit hoofdstuk wordt er gekeken naar technologieën die eventueel toepasbaar zijn op het ontwikkelproces van de nieuwe editor. Eerst zullen de huidige gebruikte technologieën op een rijtje worden gezet. Hierna zal er gekeken worden naar de toekomst van de editors. Tenslotte zal het probleem in kaart worden gebracht waarop eventuele oplossingen zullen worden geadviseerd.

3.1 Wat wordt er verstaan onder 'technology stack'

Onder een 'technology stack' (of 'tech stack') verstaan we de onderliggende bouwblokken waarop de desbetreffende applicatie gebouwd is. Deze bouwblokken bestaan uit onder andere frameworks, libraries, programmeertalen, softwareproducten en eventuele tooling[37].

3.2 Waarom is dit belangrijk?

Het is erg belangrijk om de tech stack in kaart te brengen omdat er elementaire informatie uit te halen is. De tech stack geeft inzicht in hoe het huidige product in elkaar steekt en eventuele consequenties wanneer er componenten in de stack aangepast worden. Dit is noodzakelijk voor het maken van een nieuwe editor die geïntegreerd moeten worden in een al bestaande tech stack. Door deze stap te nemen wordt er goed gekeken naar hoe de nieuwe editor in het totaal plaatje zou kunnen passen. Zo kan het voor komen dat er bepaalde software wordt gebruikt die nauw samenwerkt met de huidige editors. Dit heeft als gevolg dat de nieuwe editors deze samenwerking moeten ondersteunen.

3.3 Huidige ontwikkelomgeving

Om de ontwikkeling van narrative games te ondersteunen heeft het bedrijf een ontwikkelomgeving opgezet. Het doel van deze omgeving is om het ontwikkelingsproces inzichtelijk te maken voor verschillende disciplines en overtollig werk, zoals het opzetten van een nieuw project, te vermijden door een leeg raamwerk aan te bieden. Dit raamwerk bevat templates (het NGT), editors en programma's om de efficiëntie en collaboratie tijdens het ontwikkelproces te bevorderen.

De ontwikkelomgeving voor narrative games heeft een onderliggende tech stack die bestaat uit verschillende programmeertalen, libraries, frameworks, Integrated development environments (IDE's) en externe applicaties. Stackshare.io[1] een website waar stacks van (bekende) bedrijven

[1] https://stackshare.io/

kunnen worden ingezien, deelt deze op in de volgende lagen[25]:

- **Application and Data**, dit betreft onder andere programmeertalen, frameworks & libraries.
- **Utilities**, hieronder vallen analytics en eventuele hulpmiddelen.
- **DevOps**, gaat over de build, test, deploy processen en het monitoren van het product.
- **Business Tools**, omvangt bestaande oplossingen voor samenwerking en marketing.

De onderliggende tech stack die de ontwikkelomgeving van narrative games ondersteund kan uiteen worden gezet volgens deze lagen. Hierdoor kan inzicht worden verkregen in hoe de huidige ontwikkelomgeving in elkaar steekt. Het bedrijf beschrijft de narrative game: 'Mission Zhobia: Winning the Peace' als een typische narrative game. Dit spel beschikt over de volgende tech stack:

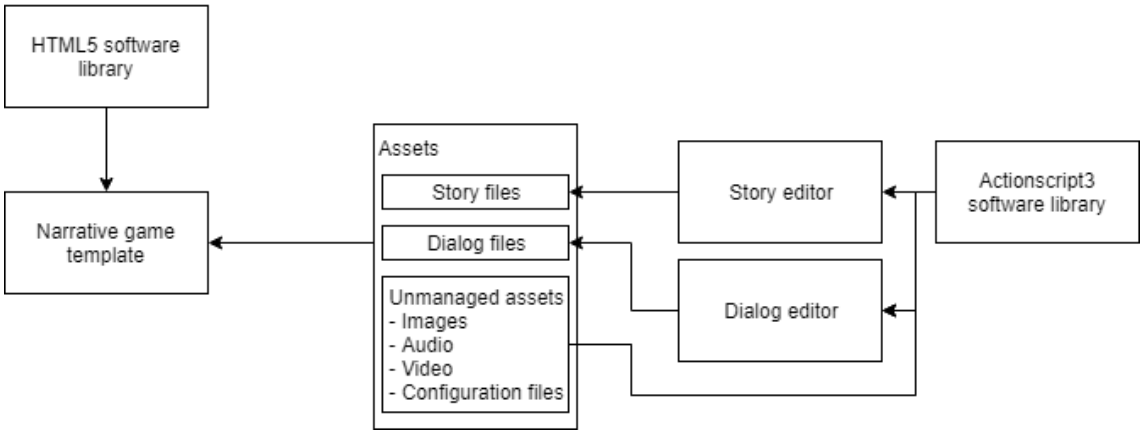
Narrative game - Technology stack	
Application and Data	
&ranj JavaScript software library	• Javascript(ECMAScript5) • CreateJS suite
Narrative game template	• Javascript(ECMAScript5) • CreateJS suite • SomaJS
&ranj ActionScript3 software library[2]	• ActionScript3
Story- & dialog editor	• ActionScript3 • Apache Flex • Flex wires
Utilities	
Analytics	• Google Analytics
DevOps	
Source control	• Beanstalk • SourceTree
Deployment	• Jenkins • SourceTree
IDE'S	• Adobe Flashbuilder[3] • Netbeans
Monitoring	• Pingdom
Business Tools	
Collaboratie	• Trello • G Suite • Slack
Documentatie	• &ranj wiki

Tabel 3.1: Huidige technology stack

Dit onderzoek betreft het opzetten van (flexibele) editors, daarom is het essentieel om de relatie tussen de huidige editors en de andere componenten binnen de 'Application and Data' laag in kaart te brengen. Hier kunnen eventuele risico's/ consequenties naar aanleiding van veranderingen

[2]Bestaat uit 3 ActionScript3 libraries gemaakt door &ranj, maar heeft één verzamelnaam
[3]Bouwt ook de editors

worden uitgelicht. Het is van belang dat de nieuwe editor goed aansluit bij de rest van de stack, zodat er onderbouwd advies op het gebied van veranderingen in de tech stack uitgebracht kan worden. De relaties tussen de componenten in de 'Application and Data' laag zijn weergeven in figuur 3.1.



Figuur 3.1: Story- en dialog editor relaties

3.4 Toekomst van de editors

De toekomst van de editors zal, zoals in de inleiding beschreven, bestaan uit een web omgeving waarin meerdere personen tegelijk kunnen werken (collaborative editing). Hierin worden aanpassingen direct getoond aan teamleden wat het mogelijk maakt om met elkaar samen te werken aan dezelfde bestanden. Deze feature wordt nog interessanter wanneer klanten bij het ontwikkelproces betrokken worden. Zij zouden dan directe feedback of zelfs aanpassingen kunnen doen aan de game content. Het bedrijf geeft aan dat ze hier in de toekomst naar toe gaan werken, maar dat er meer onderzoek en resources nodig zijn om dit mogelijk te maken.

Hoewel dit vraagstuk niet in dit onderzoek past kan er wel rekening mee worden gehouden. Ook al zal het bedrijf voorlopig gebruik maken van een desktopapplicatie, moet er wel een blik op de toekomst geworpen worden. Bij het opzetten van een nieuwe editor vindt &ranj toekomstgerichtheid een belangrijk aspect; het bedrijf wil dan ook niet opnieuw een hele editor opzetten. Bij de keuzes die invloed hebben op de tech stack zal hier dan ook op worden gelet.

3.5 Probleem

Beschikken over een stabiele en toekomstig gerichte tech stack is cruciaal voor een succesvol product. De tech stack heeft een directe invloed op de toegankelijkheid, schaalbaarheid en toekomstgerichtheid van de toekomstige editor.

3.5.1 Onzekerheid in de toekomst

De huidige editors zijn gemaakt in Apache Flex. Dit is een omgeving waarin applicaties gemaakt kunnen worden met de hulp van ActionScript3 om logica te kunnen programmeren en MXML wat het definiëren van lay-outs toelaat in een XML-formaat[40]. Projecten kunnen gecompileerd worden naar SWF-bestanden die kunnen worden uitgevoerd in een Flash- of Air run time. Vorig jaar, 25

juli, liet Adobe in een blog post weten dat ze ondersteuning voor Flash gaan beëindigen in 2020[38]. Hiernaast lijken er ook weinig updates plaats te vinden. Volgens de website van Apache Flex was de laatste update op 22 november 2017[2]. Tenslotte werd Flash al eerder in meerdere populaire browsers standaard geblokkeerd vanwege veiligheidsredenen[36]. Dit gaat tegen het toekomstbeeld van de editors in, het bedrijf wilt toewerken naar een webapplicatie. Verder leidt dit alles naar een onzekere toekomst van Apache Flex.

3.5.2 Kleine community

Het aanbod van libraries, klare oplossingen op veelvoorkomende problemen, is naar verhouding vrij minimaal omdat de community rond Apache Flex en ActionScript3 in vergelijking tot andere ontwikkelomgevingen relatief klein is. In de 'populaire technologieën' sectie van de enquête die Stack Overflow jaarlijks afneemt zijn Apache Flex en ActionScript3 niet te vinden[49]. Ondanks dat de Apache Flex community wel op Stack Overflow zit[20]. Een kleinere community kan leiden tot minder hulp en een gebrek aan oplossingen voor veel voorkomende problemen. Dit is ook terug te zien aan 'Flex Wires', een library die de editors gebruiken om de nodes met lijnen aan elkaar te verbinden. De library is slecht aanpasbaar en biedt weinig functionaliteit. Verbindingen kunnen niet aangepast worden, er verschijnt altijd een grijze kromme lijn. Dit heeft als gevolg dat bepaalde features niet haalbaar zijn in de huidige editors. Hiernaast zitten er fouten in Flex Wires die alleen opgelost kunnen worden in de library zelf. Zo kunnen de verbindingen uit het niks verdwijnen, waardoor niet meer te zien is welke relaties er bestaan tussen nodes.

3.5.3 Overtollige libraries

Zowel de editors als het NGT maken gebruik van de &ranj software library welke generieke functionaliteiten en datastructuren bevat (zie figuur 3.1). Echter moeten er twee libraries onderhouden worden omdat de huidige editors en het NGT geschreven zijn in verschillende programmeertalen. Het implementeren van nieuwe generieke functionaliteiten moet dubbel gedaan worden en de libraries moeten beide up-to-date zijn, omdat het NGT en de editors anders mogelijk niet meer goed samen kunnen werken.

3.6 Ontwikkelplatformen

Het huidige ontwikkelplatform, Apache Flex, heeft een onzekere toekomst en sluit niet aan bij het toekomstbeeld van de editors die het bedrijf heeft. Het zou zonde zijn om te prototypen in een ontwikkelomgeving die niet gericht op de toekomst is.

3.6.1 Aandachtspunten

Bij het zoeken naar een gepast ontwikkelplatform werd er gelet op: de community om het platform heen, de toekomstgerichtheid van het platform en hoeveel werk het gaat kosten om deze te integreren in de huidige tech stack. Deze aspecten zijn gesorteerd op belang van hoog naar laag en worden hieronder vermeld met concrete vragen:

1. Community
    - Bestaat er een actieve community waarin mensen elkaar verder helpen met problemen?
    - Zijn er bestaande oplossingen voor een visual scripting interface?

2. Toekomstgerichtheid
    - Door wie wordt het ontwikkelplatform onderhouden?
    - Hoe ziet de toekomst van het ontwikkelplatform eruit?
    - Kan er naast een desktopapplicatie ook uitgerold worden naar een web omgeving?
3. Integratie
    - Sluit het ontwikkelplatform aan bij de rest van de tech stack?
    - Sluit het ontwikkelplatform aan bij het bedrijf? Kunnen programmeurs overweg met het platform, zo niet hoe stijl is de leercurve?

3.6.2 Selectie

Er is een selectie gemaakt uit populaire en mogelijk passende ontwikkelplatformen:

1. Haxe
2. Electron
3. NW
4. Qt

Verder zullen de volgende ontwikkelplatformen kort worden behandeld, omdat deze potentie hadden maar al gauw niet de oplossing bleken te zijn.

1. Unity
2. Apache FlexJS

Unity

Het bedrijf wilt gebruik gaan maken van Unity voor de ontwikkeling van narrative games. Unity biedt een platform waarmee de verschillende disciplines in een projectteam beter samen kunnen werken. Het integreren van de nieuwe editor met Unity kan voordelen met zich meebrengen, zoals beter feedback en een betere workflow. Echter is het niet aan te raden om een gehele editor in Unity te maken. Unity is van origine een game engine. Er kunnen simpele extensies gemaakt worden die getoond kunnen worden in Unity, maar een gehele narrative editor maken als Unity extensies vereist veel tijd. Hiernaast moeten extensies gemaakt worden op een manier die Unity afdwingt. Naast dat dit de oorzaak is van de grote hoeveelheid vereiste tijd brengt het ook limitaties met zich mee. Als de Unity API niet beschikt over een bepaalde benodigde functionaliteit is het niet mogelijk of gaat het erg veel tijd en creativiteit kosten. Tenslotte heeft dit zware consequenties op de toekomst van de editor. Mocht het bedrijf ooit beslissen om van Unity weg te stappen dan zullen ze een deel van de editor opnieuw moeten ontwikkelen, omdat een groot deel van de code specifiek voor Unity geschreven zal zijn. Idealiter wilt het bedrijf niet afhankelijk zijn van Unity.

Apache FlexJS

Met de val van Flash is Apache een oplossing gaan zoeken om projecten van Apache Flex te compileren naar JavaScript. De oplossing die Apache heeft ontwikkeld heet Apache FlexJS, een variant op Apache Flex die code omzet naar JavaScript[12]. Als &ranj besluit componenten van de huidige editors te hergebruiken voor de nieuwe editors is het een optie om gebruik te maken van Apache FlexJS. De nieuwe oplossing van Apache is echter nog niet getest in grotere applicaties en op de download pagina laat Apache weten dat er aardig wat features missen en bugs in zitten: "The Apache Flex team is pleased to offer this release, available as of 27 June 2017. Expect lots of bugs and missing features."[3]. De laatste update was op 27 juni 2017, wat alweer bijna een jaar geleden is. Tenslotte lijkt een groot deel van de web community al weg gestapt te zijn van Apache Flex en ActionScript3. Mogelijk omdat Apache niet snel genoeg met een oplossing kwam

Haxe

Haxe, een project dat gestart is op 22 oktober 2005, is een omgeving waarin applicaties geprogrammeerd kunnen worden in de Haxe programmeertaal. Deze programmeertaal is object georiënteerd, 'strictly typed' en de syntax lijkt op een mix tussen ActionScript3 en Java. Als de logica eenmaal geprogrammeerd is kan het ontwikkelplatform trans compileren, de Haxe programmeertaal omzetten, naar 12 verschillende programmeertalen[5]. De focus van Haxe lijkt dan ook te liggen op het 'write once, run anywhere' principe.

Community Het open-source ontwikkelplatform lijkt klein maar actief. Dit blijkt uit fora en de aanwezigheid op social media[13]. Hiernaast werd er op 3 tot 5 mei een Haxe bijeenkomst georganiseerd, waarin de community samen kwam en naar meerdere (gast)sprekers luisterde over de mogelijkheden die Haxe biedt[15]. Deze bijeenkomsten, ook wel 'summits' genoemd worden gehouden sinds 2014[34], wat duidt op een vraag naar het ontwikkelplatform. Hoewel de community aardig wat oplossingen en raamwerken heeft gecreëerd voor Haxe[33] mist er wel een oplossing voor een visual scripting interface die kan helpen bij het opzetten van de editors.

Toekomstgerichtheid Het Haxe platform wordt ondersteund door een zo genaamde 'Haxe Foundation'. Deze bestaat uit donaties en betaalde ondersteuningsabonnementen. De lijst van bedrijven die de 'Haxe Foundation' financieel ondersteunen bestaat uit 6 vrijwel onbekende bedrijven. Hiernaast is de roadmap die te vinden is op de website van Haxe vrij minimaal en achterhaald. Doordat het ontwikkelplatform kan trans compileren naar 12 verschillende programmeertalen, waaronder Javascript, betekent dit wel dat er zowel desktopapplicaties als webapplicaties kunnen worden uitgerold.

Integratie De huidige tech stack komen de Javascript en Actionscript programmeertalen terug. Hoewel de Haxe programmeertaal inspiratie heeft genomen van ActionScript3 kan het wel tijd kosten om de taal te leren. Hiernaast zal er kennis vergaart moeten worden van het ontwikkelplatform zelf en er zal een nieuwe deployment pipeline opgezet moeten worden. Om de software library te kunnen gebruiken zal er een tussen laag geprogrammeerd moeten worden, zoals dit staat beschreven in de handleiding[14].

Electron

Wat begon op 15 juli 2013[31] als een project genaamd 'atom shell' die ter ondersteuning diende voor de populaire tekst bewerker genaamd 'atom editor'[4], is sinds 23 april 2015 bekend als

[4]https://atom.io/

Electron[47]. Dit raamwerk is open-source en geeft ontwikkelaars de mogelijkheid om cross-platform desktop apps te creëren met behulp van web technologieën, zoals HTML, CSS en JavaScript. Om dit alles mogelijk te maken faciliteert Electron Chromium[5] (het browser project achter de populaire Chrome browser) en NodeJS[6], een cross-platform JavaScript run-time omgeving.

Community Naast dat het project op GitHub 2.636 volgers, 59.906 favorieten en 7.844 forks[7][8] [11] heeft, weet Electron een gigantische community om zich heen te vormen door web technologieën en web ontwikkelaars te betrekken. Volgens het onderzoek naar ontwikkelaars van StackOverflow blijkt dat JavaScript, HTML en CSS de meest populaire technologieën van 2018 zijn[32]. JavaScript is al 6 jaar de meest gebruikte programmeertaal volgens de StackOverflow onderzoeken. Hiernaast wordt NodeJS het meest gebruikt van alle frameworks, libraries en tools[32]. De populariteit van Javascript en NodeJS leidt tot vele diagramming libraries die een bestaande oplossing bieden op een visual scripting interface. Deze community is precies wat &ranj zoekt. Het bedrijf zoekt naar een zogenaamde flexibele schil, waarbij de kern de werknemers zijn en de schil bestaat uit bestaande oplossingen die direct toepasbaar zijn in de werkwijze of op de producten van &ranj.

Toekomstgerichtheid Electron wordt onderhouden door GitHub[9] een platform waarop software beheerd kan worden door middel van versie beheer (git). Github zegt te beschikken over een community van 27 miljoen mensen, gemeten in maart 2018[1]. Hiernaast biedt het platform opslag voor 80 miljoen verschillende softwareprojecten. Verder wordt Electron gebruikt door bekende desktopapplicaties zoals: Skype, GitHub Desktop, Visual Studio Code, Slack en Atom[9]. Door tijdens het ontwikkelproces van de editors een duidelijke scheiding te maken tussen de applicatie en Electron kan de toekomstgerichtheid bevorderd worden. De applicatie zonder Electron blijft een webapplicatie, wat betekend dat deze relatief makkelijk omgezet kan worden naar een web omgeving. Dit sluit goed aan bij de toekomstvisie van &ranj besproken in paragraaf 3.4.

Integratie In de huidige tech stack wordt er veel gewerkt met JavaScript. Een groot probleem, zoals eerder, besproken zijn de overtollige software libraries. Deze libraries bevatten herbruikbare componenten voor zowel het NGT als de huidige editors. Echter zijn het NGT en de huidige editor ontwikkeld in verschillende programmeertalen, JavaScript en ActionScript3, waardoor er 2 verschillende versie bijgehouden moeten worden. Het overstappen van ActionScript naar JavaScript kost wat werk, maar omdat de syntax van ActionScript geïnspireerd is door Javascript zal het proces soepeler kunnen verlopen. Door over te stappen naar Electron en ActionScript uit te sluiten hoeft de ActionScript library van &ranj niet meer onderhouden te worden; er kan gebruik worden gemaakt van de JavaScript software library. Herbruikbare componenten bevinden zich hierdoor dan in één library.

NW

NW.js, eerder bekend als 'node-webkit', is een open source run-time gebaseerd op Chromium en NodeJS. Het biedt de mogelijkheid om NodeJS modules direct aan te roepen in HTML-bestanden. Deze run time is erg vergelijkbaar met Electron in de zin dat ze beide een relaties hebben tot Chromium en NodeJS. Community Het GitHub project van NW.js heeft op het moment[10] 1.812 volgers, 33.689 favorieten en 3.745 forks[24]. Om de community samen de brengen heeft NW.js

[5] https://www.chromium.org/

[6] https://nodejs.org/

[7] Een 'fork' is een copy van andermans project en wordt meestal gebruikt als startpunt van eventuele uitbreidingen en aanpassingen.

[8] Gemeten op: 11-05-2018 17:09

[9] https://github.com/

[10] Gemeten op: 11-05-2018 17:09

een Gitter[11], wat fungeert als een chatroom, opgezet waarin de community met elkaar kan praten en elkaar verder kan helpen. Hiernaast lijkt de community aanwezig te zijn op StackOverflow[21]. Vanwege de grote community rondom web development met onder andere Javascript als veelgebruikte programmeertaal bestaan er genoeg libraries waarmee een visual scripting interface opgezet kan worden.

Toekomstgerichtheid NW.js wordt gesponsord door Intel[12], maar uit data van Github[13] blijkt dat er vooral één persoon actief aan werkt. Het project blijkt slow but steady uitgebreid te worden. Er zijn een groot aantal applicaties gemaakt met NW.js[19]. De meest bekende is misschien wel de WhatsApp desktopapplicatie. Echter kwam deze applicatie ook naar boven in de lijst van Electron applicaties. Na de Whatsapp desktopapplicatie gedownload te hebben blijkt dat deze Electron gebruikt, wat te zien is aan de bestand structuur van de applicatie en het overduidelijke 'electron.asar' bestand. Een NW.js applicatie is net zoals Electron een browser in een desktopapplicatie. Als er tijdens het ontwikkelingsproces een duidelijke scheiding wordt gelegd tussen de applicatie zelf en NW.js kan dezelfde met relatief kleine moeite ook ingezet worden als webapplicatie. Ook dit slaat goed aan bij de toekomst van de editors zoals beschreven in paragraaf 3.4.

Integratie Net zoals Electron zal NW de ActionScript3 library overbodig maken, waardoor alleen nog de JavaScript library onderhouden hoeft te worden. Het overstappen zal wat werk kosten, maar relatief makkelijk zijn vanwege de overeenkomsten tussen de ActionScript3 en JavaScript syntax. Programmeurs binnen het bedrijf zijn meer bekend met JavaScript dan ActionScript3 wat het overstappen naar JavaScript makkelijker kan maken voor de ontwikkelaars.

Qt

Qt is een raamwerk waarin crossplatform applicaties kunnen worden ontwikkeld. Hiernaast biedt het raamwerk een manier om graphical user interfaces (GUI) op te zetten[27]. Qt is geschreven in C++, maar ontwikkelaars kunnen ook gebruik maken van andere programmeertalen[18]. Echter wordt er aangeraden om in C++ te ontwikkelen.

Community De Qt community is actief op StackOverflow[22] en het Qt forum[16]. Vooral op het Qt forum is de community actief. Hiernaast organiseert Qt jaarlijkse summits en Qt dagen. Er zijn geen libraries gevonden die kunnen helpen bij het opzetten van de visual scripting interface. Wel heeft Qt een voorbeeldje opgezet waarmee dit eventueel bereikt zou kunnen worden[7]. Dit neemt echter niet weg dat het veel werk zal gaan kosten.

Toekomstgerichtheid Qt wordt ontwikkeld en onderhouden door het bedrijf zelf en biedt een open source en commerciële versie van het raamwerk[8]. Verder maken bekende bedrijven zoals Valve[26], Blizzard[4], VideoLan[29] en AMD[28] gebruik van Qt. Wat er op duidt dat Qt over een goed getest ontwikkelplatform beschikt. Er is een mogelijkheid om webapplicaties te ontwikkelen in Qt[30][6][35]. Echter is het niet duidelijk of dezelfde codebase gebruikt kan worden voor zowel web- als desktopapplicatie.

Integratie Als raamwerk geschreven in c++ past het minder goed bij een tech stack die vooral bestaat uit web technologieën. Hiernaast is het voor de editors lastig om voordeel te halen uit een low level programmeertaal zoals c++, omdat deze de fijne controle die c++ biedt niet benutten. De editors profiteren niet van kleine beetjes extra prestatie op het gebied van snelheid, omdat het slecht een tool is; het spel wordt uiteindelijk gebouwd in het NGT. De huidige tech stack die alleen bestaat uit high level programmeertalen resulteert in programmeurs die hier goed mee op

[11] https://gitter.im/nwjs/nw.js
[12] https://www.intel.com/
[13] https://github.com/nwjs/nw.js/graphs/contributors

weg kunnen. Om vervolgens een low level programmeertaal, zoals c++, te introduceren kan lastig zijn zonder programmeurs met ervaring.

NW vs Electron

NW en Electron lijken beide hetzelfde doel te delen: desktopapplicaties ontwikkelen in HTML, CSS en JavaScript. Echter blijkt Electron de populairdere optie te zijn van de twee. Dit blijkt uit de statistieken van GitHub (zie tabel 3.2). Hiernaast laat een analyse tool genaamd "IS IT MAINTAINED?"[14] zien dat er een significant verschil zit in de snelheid waarop vraagstukken van gebruikers beantwoord worden.

GitHub statistieken	**NW**	**Electron**
Volgers (Watches)	1.812	2.636
Favorieten (Stars)	33.689	59.906
Forks	3.745	7.844
Bijdragers (Contributors)	98	751
Gemiddelde oplossingstijd van gestelde vraagstukken	5 dagen	22 uur
Open vraagstukken	29%	5%

Tabel 3.2: NW vs Electron (Gemeten op: 11-05-2018 17:09)

Zowel NW als Electron kunnen worden beschouwd als battle tested[19][10]. Hiermee wordt bedoeld dat er meerdere applicaties bestaan die ontwikkeld zijn in deze ontwikkelplatformen. Echter zijn de meeste applicaties ontwikkeld in Electron, voorbeelden hiervan zijn: 'Visual Studio Code'[15], 'Slack'[16], 'Atom'[17] en 'Discord'[18]. In het gebruik van de ontwikkelplatformen zit er naast de API ook een verschil in het entry point. Beide definiëren het ingangspunt van de applicatie in het package.json bestand. In NW kan dit zowel een HTML-bestand als een JavaScript bestand zijn. Electron dwingt het gebruik van een JavaScript bestand af om meer controle te bieden over het frame waarin de applicatie zich bevindt. Tenslotte viel er iets op aan de statistieken van GitHub. Hieraan is te zien dat een persoon met de gebruikersnaam "zcbenz" tussen 2012 en 2013 relatief actief was op het NW-project. Na deze periode is deze persoon, Cheng Zhao, gaan werken aan Electron. Cheng Zhao heeft gewerkt aan NW tijdens zijn stageperiode. Hij beschrijft Electron als een tweede poging op NW[50].

3.7 Conclusie en aanbeveling

De mogelijkheid om desktopapplicaties te kunnen ontwikkelen door middel van web technologieën is ideaal voor het bedrijf. Het sluit goed aan bij de toekomst van de editors, omdat deze met minimale aanpassingen in een web omgeving kunnen worden geplaatst. Verder biedt de community rondom JavaScript oplossingen voor de interface van de editors. De hoeveelheden JavaScript oplossingen is precies wat &ranj zoekt. Het sluit aan bij het concept van de 'flexibele schil', waarbij de schil de community en haar oplossingen zijn en de medewerkers van &ranj de kern vormen. Tenslotte past een JavaScript raamwerk goed in de huidige tech stack, omdat dit het makkelijker maakt om met de JavaScript software library van &ranj te communiceren. Daarnaast werkt het bedrijf vaak

[14] http://isitmaintained.com
[15] https://code.visualstudio.com/
[16] https://slack.com/
[17] https://atom.io/
[18] https://discordapp.com/

met JavaScript waardoor programmeurs bekend zijn met de programmeertaal. Zowel Electron als NW laten het ontwikkelen van desktopapplicaties met behulp van web technologieën toe. Echter is de community rondom Electron groter, wat mogelijk komt door de ondersteuning van GitHub. Hiernaast biedt de ondersteuning van GitHub en het aantal populaire applicaties de zekerheid dat Electron voorlopig zal blijven bestaan. Tenslotte lost Electron sneller vraagstukken van gebruikers op. Dit neemt niet weg dat NW niet zou kunnen werken in deze context. Het is niet rechtvaardig om dit ontwikkelplatform compleet af te schrijven. Om zeker te zijn van een juiste keuze zou er een demo gemaakt kunnen worden in zowel NW als Electron. Beide zullen hoogstwaarschijnlijk geen limitatie stellen aan de editor, omdat deze weinig gebruik maakt van native APIs. Vanwege het gebonden tijdslimiet aan dit onderzoek zal er gekozen worden voor Electron. Deze keuze is gemaakt op het gebied van community en toekomstgerichtheid; Electron heeft een grotere community om zich heen en met de ondersteuning van GitHub zal het product voorlopig blijven bestaan.

3.8 Opzetten van de user interface

De user interface (UI) synchroon houden met de achterliggende staat kan erg rommelig en lastig zijn in standaard HTML en JavaScript. Code wordt al gauw onleesbaar en bij een kleine verandering in de staat van de applicatie wordt heel de UI geüpdatet. Om dit probleem op te lossen hebben meerdere bedrijven JavaScript UI libraries en raamwerken opgezet. Deze oplossingen delen de UI op in componenten waarin zich component specifieke logica bevindt; componenten bevorderen de encapsulatie van logica. Verder kunnen deze componenten, als de logica erachter goed ingekapseld is, in andere componenten gebruikt worden. Dit resulteert in een manier om een houdbare en flexibele UI op te zetten.

3.8.1 User interface editors

De UI van de huidige editors kan worden op gedeeld in componenten met ieder haar eigen functionaliteit. Al deze UI-componenten bevinden zich op één pagina. De story- als dialog editor lijken te beschikken over vrijwel dezelfde (hoofd)componenten:

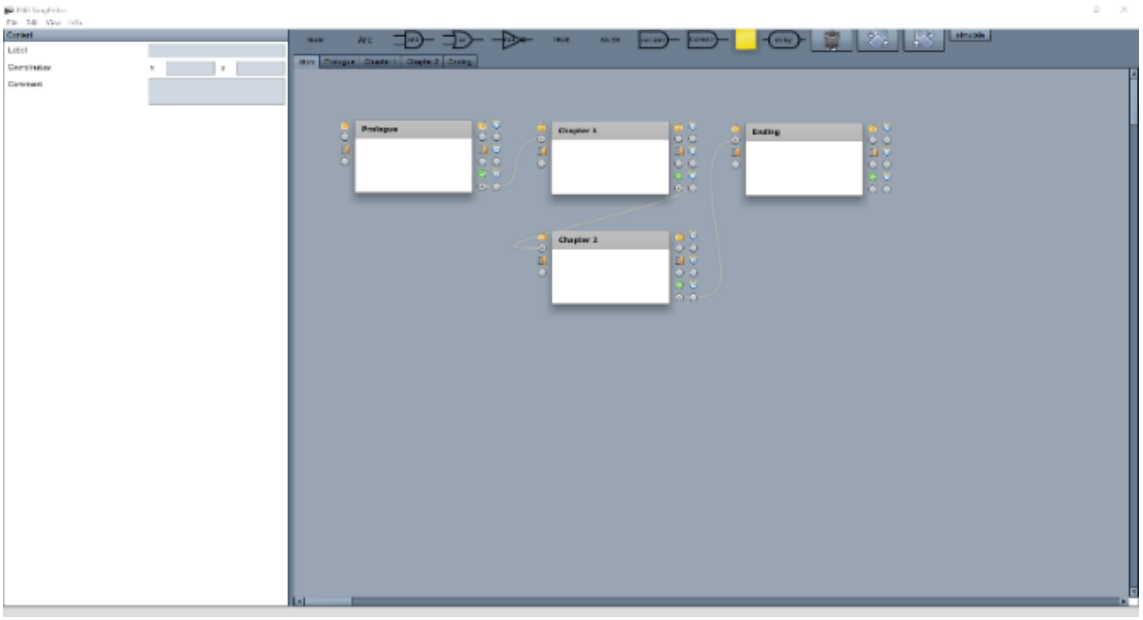
1. Inspector
2. Toolkit
3. Tabs
    (a) Canvas

3.9 User interface frameworks & libraries

Er zijn meerdere oplossingen beschikbaar voor het maken van user interfaces. Sommige bestaan uit een compleet raamwerk en een dwingen een bepaalde manier van werken af. Er kan gezegd worden dat deze een grote eigenzinnigheid hebben. Andere oplossingen zijn kleine libraries die zich alleen focussen op het inkapselen van componenten.

3.9.1 Aandachtspunten

Bij het zoeken naar een passende oplossing wordt er gekeken naar de volgende punten:



Figuur 3.2: Story Editor interface

1. De omvang van het ecosysteem
2. Eigenzinnigheid
3. Toekomstgerichtheid
    - Wie onderhoudt/ steunt het project?
    - Welke (populaire) producten gebruiken deze oplossing?
4. Snelheid
5. Leercurve

Verwachtingen

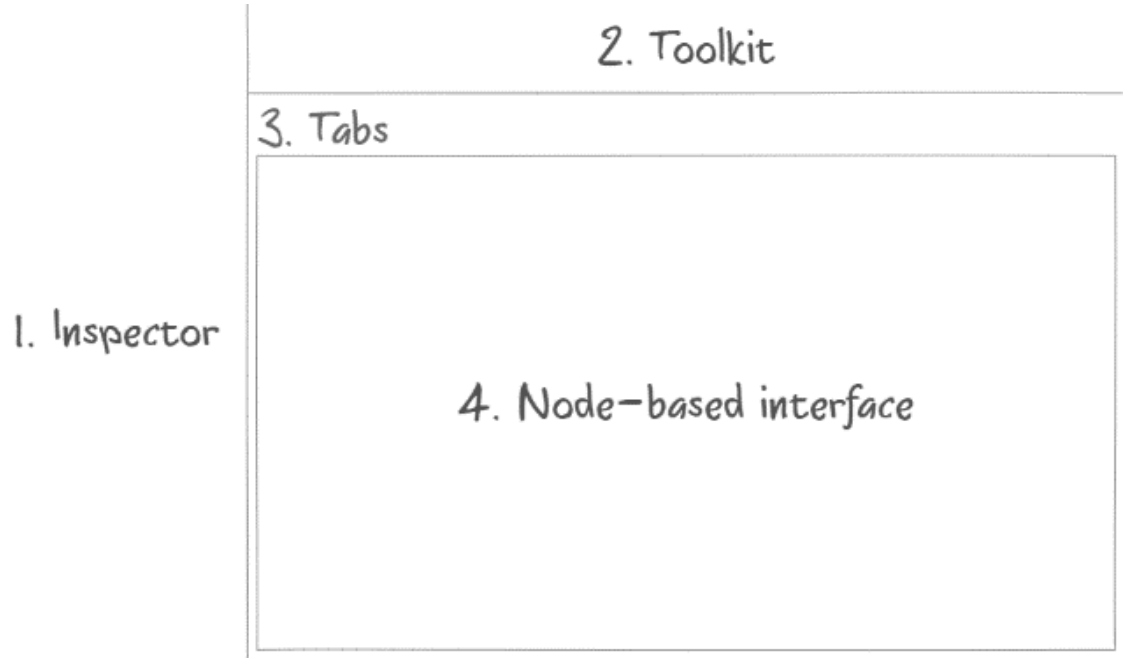
Er wordt gezocht naar een oplossing waarmee componenten in gekapseld kunnen worden. Een ecosysteem met meerdere doeleinden kan worden beschouwd als overbodig. Zo bestaan de huidige editors uit één scherm wat een router[19] overbodig maakt. Daarom zou een ecosysteem met een router redundant zijn.

Omdat er slechts gezocht wordt naar een component gebaseerde oplossing is het gewenst dat het raamwerk of de library vrijwel niet eigenzinnig is.

Verder is het belangrijk dat de ontwikkeling van het product actief is en dat deze niet snel zal verdwijnen. Een voorspelling kan gedaan worden op basis van (populaire) applicaties die het product gebruiken en of er (grote) bedrijven zijn die het product onderhouden of steunen.

Tenslotte zijn de snelheid en leercurve van het product minder belangrijk voor dit probleem, mits deze niet enorm afwijken van de rest. Bij de editor zal flexibiliteit en toekomstgerichtheid boven snelheid gaan. Hiernaast zal het ontwikkelen van een nieuwe editor moeten resulteren in een toekomstgerichte oplossing die nog voor jaren gebruik zal kunnen worden. Dit maakt het minder erg als de leercurve van het product iets hoger ligt.

[19]Een klasse die verantwoordelijk voor navigatie binnen een applicatie



Figuur 3.3: Editor componenten

3.9.2 Selectie

Om de staat van de applicatie synchroon te houden met de UI kan er gebruik worden gemaakt van bestaande oplossingen. Er is een selectie gemaakt uit populaire user interface raamwerken en libraries:

- Vue
- Angular
- React
- Ember

3.9.3 Resultaten

De geselecteerde oplossingen zijn geanalyseerd op de eerdergenoemde aandachtspunten. Het resultaat wordt hieronder weergeven:

Virtual DOM

In het verleden waren webapplicaties statisch en relatief klein. Tegenwoordig wordt er veel gewerkt met (grotere) single page applications (SPA). Een SPA bestaat uit kleine individuele componenten die met elkaar kunnen communiceren en los van elkaar geüpdatet of vervangen kunnen worden[42].

[20] https://vuejs.org/
[21] https://angular.io/
[22] https://reactjs.org/
[23] https://www.emberjs.com/
[24] TypeScript is een getypeerde 'superset' van JavaScript. Geschreven TypeScript code compileert naar JavaScript.
[25] Command line interface tooling
[26] http://www.stefankrause.net/js-frameworks-benchmark7/table.html

	Vue[20]	**Angular**[21]
GitHub stats (05/03/2018)		
Volgers	4.561	2.918
Favorieten	85.500	33.694
Forks	12.545	8.275
Gebacked door	1 persoon, Yuxi (Evan) You	Google
"Native"language	JavaScript	Typescript[24]
TypeScript ondersteuning	Ja, er zijn typing beschikbaar	Ja, out of the bo
Ecosysteem	Modulair, o.a. router, state management, cli, RxJS integration	Modulair, o.a. r
Leercurve	Makkelijk, door de flexibiliteit die Vue biedt	Lastiger door de
Snelheid[26]	Snel, Virtual DOM	Snel
Eigenzinnigheid	Klein	Groot, je moet i

De webpagina wordt nooit in zijn geheel ververst of herladen. Hiernaast is een SPA verantwoordelijk voor het correct afhandelen van gebruikersinvoer. Webapplicaties moet constant het document object model (DOM) achter de webpagina veranderen om feedback te kunnen tonen aan de gebruiker nadat deze input heeft geleverd. Voor grotere SPA-webapplicaties is dit een probleem, omdat DOM bestaat uit een boom. De nodes van de DOM boom zijn makkelijk af te gaan, maar in een grotere boom structuur (zoals in grote SPA applicaties) wordt dit al gauw een langdurig proces. Hier komt virtual DOM in het spel. Dit is een abstractie van de 'traditionele' DOM[45]. Veranderingen in de virtual DOM resulteren in de executie van een verschil algoritme. Het resultaat van dit algoritme wordt vervolgens gereflecteerd in de "echte" DOM, waarin alleen de veranderde nodes aangepast worden. Door delen aan te passen in plaats van de gehele DOM kan de webapplicatie sneller reflecteren op de input van de gebruiker. Zowel Vue als React maken gebruik van Virtual DOM.

TypeScript

Uit een onderzoek naar code kwaliteit op GitHub blijkt dat strongly typed programmeertalen minder gevoelig zijn voor fouten dan loosly typed programmeertalen[46]. JavaScript is een voorbeeld van een loosly typed programmeertaal. Om code kwaliteit te verhogen en menselijke fouten te voorkomen wordt er geadviseerd om gebruik te maken van een strongly typed programmeertaal. Volgens onderzoek naar code kwaliteit op GitHub worden er minder fouten gemaakt in TypeScript projecten[46]. TypeScript is een strongly typed superset van JavaScript, wat betekent dat TypeScript code gecompileerd kan worden naar JavaScript. Dit proces wordt ook wel transpiling genoemd.

TypeScript is ontwikkeld en wordt onderhouden door Microsoft. Hiernaast lijkt bijna iedere JavaScript library ondersteuning te bieden door typings aan te bieden. Typings zijn bestanden die TypeScript gebruikt om onderscheid te maken tussen verschillende types. Via deze typings kunnen JavaScript libraries samen werken met TypeScript code. Er is verder geen onderzoek gedaan naar alternatieven naast TypeScript, zo zou Flow[27] ook een oplossing kunnen zijn.

3.9.4 Conclusie en aanbeveling

Uit eerder beschreven verwachtingen blijkt dat er gezocht wordt naar een kleine library die zich bezighoudt met het inkapselen van componenten. Alle vier de oplossingen beschikken over een

[27] https://flow.org/

modulair ecosysteem, maar React steekt hier bovenuit. React is een library wat betekend dat deze een relatief kleine omvang heeft in tegenstelling tot de andere oplossingen die gezien worden als frameworks. De oplossing van Facebook, React, regelt de synchronisatie tussen ingekapselde componenten en de staat van de applicatie.

Vue is een opkomende technologie die naast React ook een potentiele oplossing kan zijn. Net zoals React maakt Vue gebruik van virtual DOM, wat voor beide een positief effect op de snelheid waarmee componenten geüpdatet en vervangen worden. Hiernaast dwingen ze allebei geen strikte werkwijze af; ze kennen allebei een kleine eigenzinnigheid. Het ecosysteem van vue biedt zelf meer "out of the box" functionaliteiten en heeft een bredere focus. Ondanks dat React zich alleen bezighoudt componenten heeft de community rondom React een eigen ecosysteem om React heen gebouwd. React is in deze context een beter passende oplossing, omdat er gezocht wordt naar een relatief kleine library die het mogelijk maakt om componenten te kunnen definiëren. React heeft een nauwe focus waar Vue een bredere focus heeft. Hiernaast wordt React gebruikt in producten van Facebook, wat betekend dat het zichzelf bewezen heeft. Vue is wat nieuwer en ondanks de toenemende populariteit wordt het minder gebruikt.

3.10 Opzetten van de visual scripting interface

Het doel van zowel de huidige als de nieuwe editor is om niet-programmeurs de game content aan te kunnen laten passen. Door gebruik te maken van een visual scripting interface kan de game content aangepast worden zonder enige programmeerkennis[**?**]. Stukken game content worden gezien als objecten met een visuele weergave die ook wel "nodes" genoemd worden. De eigenschappen van deze objecten kunnen vervolgens aanpast worden in de editor zelf.

Door nodes met elkaar te verbinden kan de gebruiker zonder een enkele lijn code het narratief moduleren.

3.10.1 Diagramming libraries

Visual scripting wordt toegepast op verschillende platformen[**?**][23][**?**]. Het wiel hoeft niet opnieuw uitgevonden te worden; bestaande oplossingen kunnen gebruikt worden voor het visualiseren van de nodes. Deze oplossingen gaan onder de naam 'diagramming libraries' omdat ze meestal gebruikt worden voor het tekenen van diagrammen. Er kan gebruik gemaakt worden van deze libraries om nodes te visualiseren.

3.10.2 Eisen

Voordat er een selectie aan libraries is gemaakt zijn er eisen opgesteld. Deze zijn gebaseerd op het MoSCoW principe. De eisen zijn verdeeld onder de volgende categorieën:

- **M**ust have; randvoorwaarden en functionele eisen, vereist voor het product
- **S**hould have; operationele eisen met groot belang, behoeftes van het product
- **C**ould have; operationele eisen, behoeftes van het product
- **W**ould have; ontwerpbeperkingen, principes en (code)kwaliteitsbewaking

Must have; de library moet

- Voorzien zijn van documentatie
- Restricties kunnen leggen op connecties tussen porten
- Nodes in elkaar kunnen voegen
- Nodes kunnen highlighten
- Controls (zoals input velden) op nodes kunnen leggen

Should have; het is van belang dat de library

- Nodes kunnen copy/ pasten*
- Acties ongedaan kunnen maken*
- Een navigeerbaar canvas aanbieden**
- Group selectie mogelijk maakt*
- De mogelijkheid biedt om grafen in grafen in te kunnen maken (sub-graphs).

Could have; het is mooi meegenomen als de library

- Een minimap kan tonen waarop alle nodes zichtbaar zijn
- Real-time collaboratie toelaat
- De nodes kan ordenen
- De nodes snapt op een raster
- Een zoekfunctie biedt
- Een verschil algoritme implementeert
- Het groeperen van nodes mogelijk maakt

Would have; om de houdbaarheid van het product te verhogen moeten de library

- Typings aanbieden

** zou eventueel zelf met relatief weinig moeite geïmplementeerd kunnen worden ** er kan gebruik gemaakt worden van andere oplossingen voor dit probleem*[28]

Selectie en analyse

Er is een selectie gemaakt uit JavaScript diagramming libraries:

- D3-node-editor[29]
- wcPlay[30]

[28] https://github.com/ariutta/svg-pan-zoom
[29] https://github.com/Ni55aN/d3-node-editor
[30] https://github.com/WebCabin/wcPlay

- JointJS[31]
- JointJS + Rappid (commerciële versie van JointJS)[32]
- MxGraph[33]
- GoJS[34]

De selectie aan diagramming libraries is afgezet tegen de opgestelde eisen en op genomen in de onderstaande matrix.

3.10.3 Conclusie en aanbeveling

Uit de analyse blijkt dat 'D3-node-editor' en 'wcPlay' niet voldoen aan de randvoorwaarden. Dit lag vooral aan de matige documentatie die de libraries meeleveren. Hierdoor is het erg lastig om de libraries toe te passen.

MxGraph en Rappid beschikken beide over een grote feature set die voldoen aan de requirements die vooraf opgesteld zijn. MxGraph was voorheen een commerciële oplossing, maar is nu gratis te gebruiken. Rappid met JointJS is de betaalde optie met collaborative editing functionaliteit. Hiernaast lijkt de community rondom JointJS groter; de library lijkt populairder te zijn dan MxGraph.

De commerciële optie, Rappid met JointJS, komt met betere documentatie en mogelijke ondersteuning. Dit kan essentieel zijn bij het opzetten van de editors. De collaborative editing functionaliteit van Rappid zou eventueel ook interessant kunnen zijn voor het bedrijf.

Omdat JointJS aan de randvoorwaarden voldoet en gratis is zal het ondersteunende prototype hiervan gebruik maken. Rappid is een laag bovenop JointJS en zou eventueel later toegevoegd kunnen worden.

3.11 Tech stack conclusie

Dit hoofdstuk zijn de risico's van de huidige tech stack in kaart gebracht. Er is geadviseerd om weg te stappen van Apache Flex vanwege haar onzekere toekomst en kleine community, waardoor er een gebrek aan bestaande oplossingen bestaat. Electron en React bieden een battle tested platform met een grotere community die gebruikt kan worden als basis voor de editor. Hiernaast komt de verouderde AS3 software library te vervallen waardoor er nog maar één software library onderhouden hoeft te worden.

De web development community biedt vele oplossingen op visual scripting interfaces. Deze diagramming libraries nemen veel werk uit handen bij het opzetten van de nieuwe editors. JointJS en Rappid worden aangeraden als diagramming library vanwege de documentatie, eventuele ondersteuning en collaborative editing functionaliteit.

De nieuwe tech stack ziet er als volgt uit:

[31] https://github.com/clientIO/joint
[32] https://www.jointjs.com/
[33] https://github.com/jgraph/mxgraph
[34] https://gojs.net/latest/index.html

Narrative game - Technology stack	
Application and Data	
&ranj JavaScript software library	• Javascript(ECMAScript5) • CreateJS suite
Narrative game template	• Javascript(ECMAScript5) • CreateJS suite • SomaJS
Editor(s)	• Electron • React • TypeScript • JointJS + Rappid
Utilities	
Analytics	• Google Analytics
DevOps	
Source control	• Beanstalk • SourceTree
Deployment	• Jenkins • SourceTree
IDE'S	• Netbeans
Monitoring	• Pingdom
Business Tools	
Collaboratie	• Trello • G Suite • Slack
Documentatie	• &ranj wiki

Tabel 3.3: Huidige technology stack

		D3-node-editor	wcPlay	JointJS	JointJS + Rappid	MxGraph	GoJS
	Prijs	Gratis	Gratis	Gratis	€ 1.500,00	Gratis[29]	$995
	Open source	Ja	Ja	Ja	Nee	Ja	Ja
	Laatste update	Recent	Sep 2016	Recent	Recent	Mrt 2018	Recent
	Documentatie	+-	+-	++	++	+	++
	Feel	+	--	++	++	+	+-
	GitHub stats						
	Volgers	24	3	141	?	170	224
	Favorieten	358	22	2,488	?	1,991	1,943
	Forks	44	2	554	?	589	1,095
	Bijdragers	5	1	49	?	12	3
	Features						
Must	Ports	X	X	X	X	X	X
Must	Connection restricties	X	X	X	X	X	X
Must	Node embedding			X	X	X	X
Must	Highlighting	X	X	X	X	X	X
Must	Custom controls/ properties	X	X	X	X	X	X
Should	Copy/ Paste	?	X		X	X	X
Should	Undo/ Redo	X	X		X	X	X
Should	Canvas Zooming/ Panning	X	X		X	X	X
Should	Group selectie	X	X		X	X	X
Should	Sub graphs		X	X	X	X	X
Could	Minimap			X	X		
Could	Real-time collaboratie				X		
Could	Auto layout			X	X	X	X
Could	Grid/ Line snapping			X	X	X	X
Could	Search		X				
Could	Diff	X					
Could	Node grouping	X		X	X	X	
W	Typings zijn beschikbaar	Ja	Nee	Ja	Ja	Community	Ja

Documentatie ratings	
++	Documentatie is professioneel opgezet, er wordt ingezoomd op specifieke onderdelen/ modules en hoe je deze eventueel kunt modificeren
+	Documentatie is goed aanwezig, er wordt ingezoomd op onderdelen en uitgelegd hoe je deze kunt gebruiken
+-	Documentatie is minimaal aanwezig, maar het zou genoeg kunnen zijn
-	Documentatie is te minimaal op de library goed te kunnen gebruiken
--	Documentatie is niet aanwezig

Figuur 3.4: Voor het laatst ingezien op: 16/05/2018

Hoofdstuk 4

Diversiteit in game content

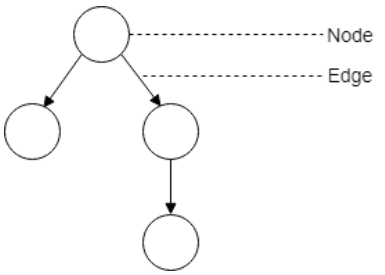
De corporate learning afdeling van &ranj houdt zich vooral bezig met het maken van narrative games op maat. Er is een ontwikkelomgeving opgezet waarin narrative games efficiënter ontwikkelt kunnen worden. In dit hoofdstuk wordt er ingezoomd op de story- en dialog editor. Dit zijn applicaties waarin het narratief achter het spel zonder enige programmeerkennis geschreven kan worden.

De huidige editors moeten telkens aangepast worden om nieuwe game content te ondersteunen die geïntroduceerd wordt door de projecten op maat. Dit hoofdstuk stelt een oplossing voor waarmee de editors op een flexibele manier om kunnen gaan met de diversiteit aan game content.

4.1 Story- en dialog editor

Tijdens het ontwikkelproces van een narrative game werken game designers, visual designers en programmeurs nauw samen. Om game designers het verhaal te laten schrijven en hierin visuals te tonen zijn er twee editors opgezet; de story- en dialog editor. De programmeurs kunnen vervolgens de game engine aanpassen om het geschreven verhaal te interpreteren, om zo het verhaal te visualiseren in het spel.

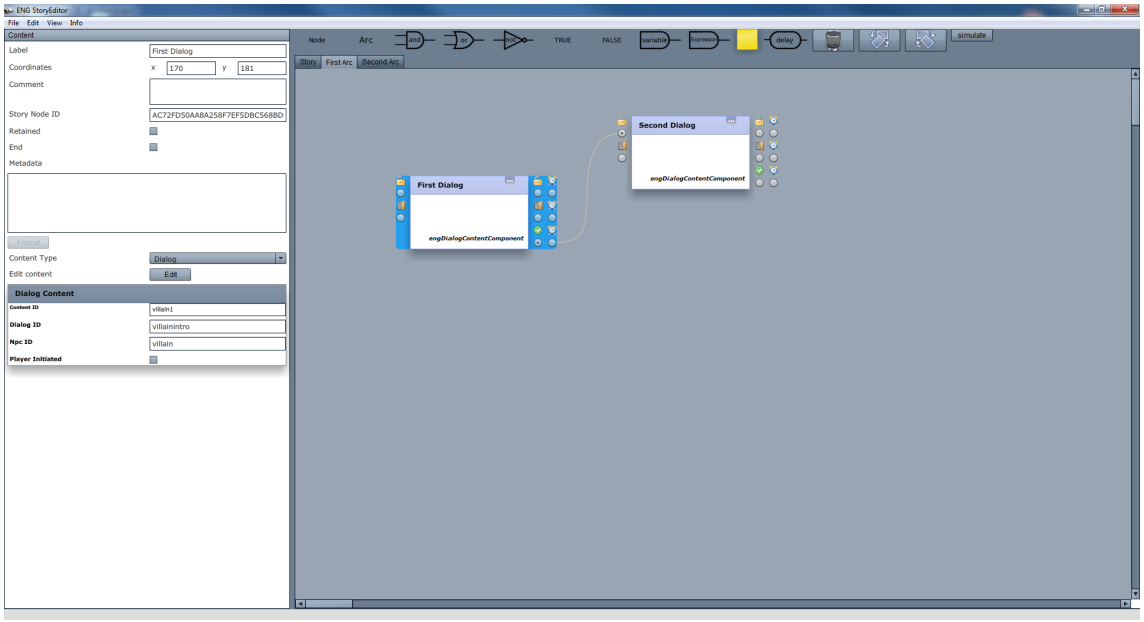
Game designers maken gebruik van de visual scripting interface die de editors bieden. Deze visual scripting interface bestaat uit visuele componenten die dienen als bouwblokken waaruit het verhaal is opgebouwd. Door deze bouwblokken op het canvas te plaatsen en met elkaar te verbinden kan er een verhaal worden geschreven. Zo zijn er bouwblokken waarmee conditionaliteit toegepast kan worden en bouwblokken die een content type vrij laten komen. Content typen zijn kleine datastructuurtjes met een betekenis in het verhaal. Een voorbeeld van zo'n content type is 'text content'. Deze bevat een tekst die getoond zal worden in het spel. In een later hoofdstuk zal er dieper worden ingegaan op content typen.



Figuur 4.1: Visuele representatie van een graaf

In de editors wordt een verhaal gerepresenteerd als een graph, in het Nederlands graaf genoemd[39]. Een graaf bestaat uit nodes en edges, zoals weergeven in figuur 4.1. De nodes zijn de bouwblokken en de edges de verbindingen tussen deze bouwblokken. Aan de editors is het goed terug te zien dat het verhaal gerepresenteerd wordt als een graaf (figuur 4.2).

In hoofdstuk 5 wordt er dieper ingegaan op het interpreteren van deze verhaal graaf.



Figuur 4.2: Een simpel verhaal gemaakt in de story editor

4.2 Content typen

Iedere game bestaat uit game content. Dit is een verzamelnaam voor alle teksten, media en mechanieken die zich binnen het spel bevinden. Om onderscheid te maken tussen game content wordt er gebruik gemaakt van een bouwblok genaamd: content node. Deze bouwblokken bevatten een content type; een stukje game content. Denk hierbij aan een dialog, tekst of afbeelding. Ieder content type heeft een eigen betekenis en doel. In figuur 4.2 wordt er gebruik gemaakt van 'dialog content' welke een dialoog zal starten. Een ander voorbeeld van een content type is 'text content', deze kan als doel hebben om tekst te tonen. Verder bevatten content types 'properties'. Dit zijn velden die verdere informatie geven over de eigenschappen van het doel. Zo heeft het content type 'text content' een property genaamd 'Text' wat de tekst is die getoond zal worden door 'text content'. Een content type is dus een datastructuur met een betekenis in de game. Door middel van content typen wordt er semantiek gebonden aan content nodes.

Door de semantische eigenschappen van een content node kunnen zowel de game designers als programmeurs onderscheid maken tussen game content. Programmeurs schrijven code om deze content types af te vangen en te interpreteren. Als er geen onderscheid bestaat weet de game engine niet wat er getoond moet worden wanneer er een content node vrijkomt.

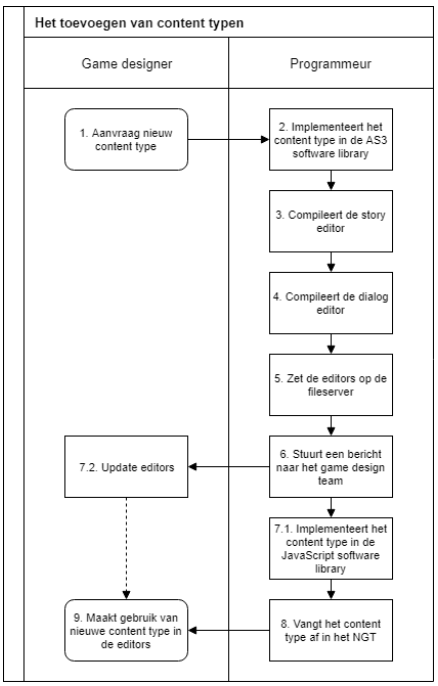
Echter hebben de meeste content typen een vrij impliciet doel. Zo kan 'text content' een tekst bevatten die uitgesproken wordt door een karakter in het verhaal, maar het kan ook een mogelijk antwoord zijn dat de speler kan geven op een vraag. Om duidelijk onderscheid te maken is het belangrijk dat content typen zo expliciet mogelijk zijn in hun doel. Dit leidt naar content typen zoals 'answer content' en 'quiz content'. Het is belangrijk dat er weinig ruimt is voor verschillende interpretaties.

4.2.1 Vervuiling van de scope

Een nadeel van content typen met een expliciet doel is dat ze meestal maar bruikbaar zijn voor een enkel project. Vooral in projecten waarin game content erg van elkaar verschild is dit een probleem, hergebruik van content typen is erg laag. Per project worden er nieuwe content types aangemaakt, omdat de game content niet overeenkomt met vorige projecten. Echter kunnen al bestaande content typen niet worden verwijderd uit de editors, omdat de editor dan niet meer gebruikt kan worden voor oudere projecten. Dit alles zorgt voor een vervuiling van de content typen scope; content typen zijn beschikbaar in project die ze niet benutten.

4.2.2 Statische definities

De editors maken gebruik van content types die gedefinieerd zijn in de &ranj software library. In de software library bevinden zich klassen die ieder een content type vormt; content types hebben een statische definitie, ze zijn in de editor gebakken. Dit biedt wel meer controle over het content type, maar deze hoeveelheid controle overtollig, content types blijven datastructuren. Hiernaast maakt dit het proces van het toevoegen en verwijderen van content types lastiger. Om dit te bereiken moet de broncode van de editor worden aangepast en vervolgens moet de editor opnieuw worden gecompileerd. Het huidige proces voor het toevoegen van een content type ziet eruit als in figuur 4.3.



Figuur 4.3: Proces: het toevoegen van content types

1. De game designer wilt onderscheid maken tussen quiz tekst die gebonden staat aan een tijdslimiet en vragen die de hoofdpersoon zichzelf stelt, waarbij de speler geen tijdsdruk heeft. Hij of zij heeft aparte content typen nodig om onderscheid te maken tussen deze twee typen game content.

2. De programmeur implementeert deze content typen in de &ranj ActionScript3 (AS3) software library. Uit deze library halen de editors de content types. In de software library wordt aan gegeven welke editors gebruik mogen maken van het content type. Hieruit kan geconcludeerd worden dat er geen encapsulatie van content typen bestaat. De editors zouden zelf aan moeten geven van welke content typen ze gebruik maken.

3. De programmeur haalt de nieuwe versie van de AS3 library binnen en past het versie nummer aan. Hij of zij compileert handmatig de story editor. Hierna wordt er “clone” achter de applicatie sleutel (application identifier) geplakt en de story editor voor de tweede keer gecompileerd. Dit maakt het openen van twee story editor schermen mogelijk. Anders kunnen er geen twee instanties van Apache Flex applicaties tegelijk openstaan.

4. Stap 3 wordt herhaald door de programmeur, maar dan voor de dialog editor.

5. De installers van de vier editors worden op de file server van &ranj gezet. Dit is een gedeelde folder waar medewerkers van het bedrijf toegang tot hebben.

6. De programmeur stuurt een mail naar het game design team waarin staat dat er een nieuwe versie

van de editors beschikbaar is. Hiernaast staan veranderingen en de locatie van de nieuwe editor in de mail.

7. Terwijl de game designers de editors updaten (7.2), implementeert de programmeur het content type in de JavaScript software library, zodat deze gebruikt kan worden in het NGT.

8. De programmeur vangt het nieuwe content type af in het NGT en linkt deze aan de correcte actie.

9. De game designer kan gebruik maken van het nieuwe content type. De game interpreteert het content type.

De programmeur moet eerst de Apache Flex editor projecten opgezet hebben en daarnaast kennis hebben van deze projecten. Hiernaast moet de editor handmatig getest worden na het toevoegen van de nieuwe content typen om te kijken of er per ongeluk niks gebroken is. Dit kan kostbare tijd en geld kosten. Een geautomatiseerde deployment pipeline zou kunnen helpen en veel werk uithanden nemen van de programmeur. Hiernaast moet het compilatie en deployment proces zoveel mogelijk vermeden worden, omdat dit tijd kost. Content typen blijven datastructuurtjes en het toevoegen of aanpassen van deze structuren zou geen compilatie van beide editors moeten vereisen.

4.2.3 Misbruik van content types

Omdat het toevoegen of aanpassen van content typen een langdradig proces is wordt dit het liefst vermeden. Vaak worden er content typen uit oudere project misbruikt om onderscheid te kunnen maken tussen game content. Zo wordt 'MinigameContent' regelmatig gebruikt voor het tonen van hele andere dingen dan minigames. Door content typen bij projecten op verschillende manieren te gebruiken kan er nooit op de eerste blik zeker gezegd worden wat voor doel het content type heeft.

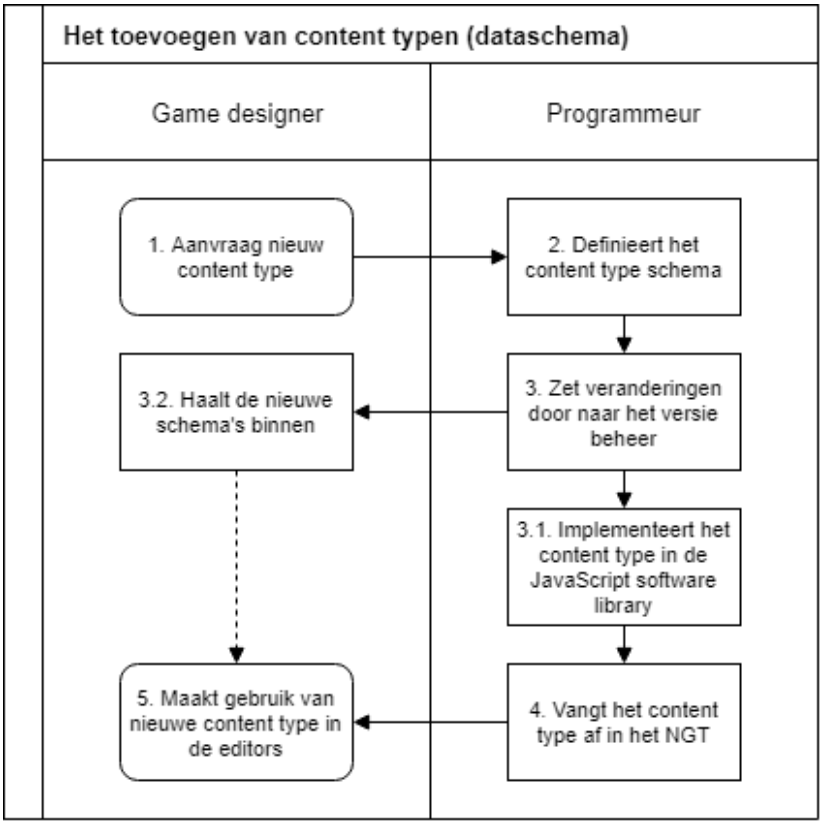
4.3 Dataschema's

Vanwege de diversiteit in game content verschillen de content typen sterk per project. De selectie aan content types wordt geacht om dynamisch te zijn. Echter staan deze statisch gedefinieerd in de &ranj AS3 software library. Er moet een manier komen om per project een selectie aan content typen te kunnen specificeren. Een content type is een (kleine) datastructuur die bestaat uit een aantal velden. Zo bestaat 'sms content', een content type die een SMS vrij laat komen, uit een afzender, datum en inhoud. Door in ActionScript3 een type aan een veld toe te kennen kan er worden afgedwongen dat afzender en inhoud een tekst zijn en dat de datum een datum is. Vervolgens verbiedt de user interface foutief gebruik; de datum moet in een correct formaat staan. Restricties opleggen aan de user interface als validatie is gevaarlijk, want het is nog steeds niet zeker of de data valide is omdat deze mogelijk ook op andere manier kan ontstaan. Er wordt dus gezocht naar een oplossing waarmee velden in een content type gespecificeerd kunnen worden. Hiernaast moeten deze velden gevalideerd kunnen worden.

Om de velden van de content typen te specificeren zal er gebruik gemaakt worden van dataschema's. Met dataschema's kan er een set aan regels worden opgelegd aan de datastructuur. Hierbij is het belangrijk dat het zowel leesbaar is voor mens als machine. Een pseudo dataschema voor 'sms content' zou eruit kunnen zien als in figuur 4.4. Dit pseudo dataschema is leesbaar voor mensen, maar niet voor computers. Computers hebben een gestandaardiseerd formaat nodig om data te kunnen interpreteren, zoals Extensible Markup Language (XML) of JavaScript Object Notation (JSON). Door gebruik te maken van een dataschema kan het proces voor het toevoegen van content typen sterk worden versimpeld. Het nieuwe proces wordt weergeven in figuur 4.5.

SMSContent bestaat uit
een **afzender** weergeven in **tekst**
een **ontvang datum** weergeven als **datum**
een **inhoud** weergeven als **tekst**

Figuur 4.4: Pseudo dataschema voor 'sms content'



Figuur 4.5: Een versimpeld proces voor het toevoegen van content typen.

XML-dataschema

XML heeft dataschema functionaliteit; de onderliggende structuur van een XML-object kan worden gespecificeerd. Dit bereikt XML door middel van elementen en attributen. De eerder beschreven 'sms content' kan worden beschreven in een XML-dataschema als in figuur 4.6. Vervolgens kunnen XML-objecten gevalideerd worden door het schema. Het XML-object in figuur 4.7 is valide, want de structuur komt overeen met die van het dataschema in figuur 4.6. JSON mist deze functionaliteit.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="smscontent">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="sender" type="xs:string" />
                <xs:element name="receiveDate" type="xs:dateTime" />
                <xs:element name="content" type="xs:string" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figuur 4.6: XML-dataschema voor 'sms content'.

```
<?xml version="1.0" encoding="UTF-8"?>
<smscontent>
    <sender>Harold</sender>
    <receiveDate>2018-04-21T11:00:00</receiveDate>
    <content>Great moves, keep it up!</content>
</smscontent>
```

Figuur 4.7: Valide XML-object van 'sms content'.

4.3.1 JSON-dataschema

JSON is een populaire keuze; het is klein en snel. Er kwam steeds meer vraag naar nieuwe functionaliteiten, zoals het kunnen specificeren van schema's[48]. Vanwege de vraag naar schema's in JSON is het er een woordenboek opgezet waarmee JSON-schema's opgezet kunnen worden[17]. In dit woordenboek staan afspraken over keywords en hun functionaliteit. Door gebruik te maken van deze keywords, met name type en properties, kan het schema voor 'sms content' wordt opgezet zoals in figuur 4.8. Het JSON-document in figuur 4.9 is valide volgens het schema in figuur 4.8.

```
{
    "type": "object",
    "properties": {
        "sender": {
            "type": "string"
        },
        "date": {
            "type": "string",
            "format": "date-time"
        },
        "content": {
            "type": "string"
        }
    }
}
```

Figuur 4.8: JSON-schema voor 'sms content'.

```
{
    "sender": "Harold",
    "date": "2018-04-21T11:00:00",
    "content": "Great moves, keep it up!"
}
```

Figuur 4.9: Valide JSON-document van 'sms content'.

4.4 Een schaalbaar dataschema voor content types

XML komt met out of the box dataschema functionaliteit, in JSON worden dataschema gemaakt op basis van afspraken. Echter zal er gebruik worden gemaakt van JSON-schema's, omdat deze beter integreert met de nieuwe tech stack. JavaScript zelf komt direct met JSON ondersteuning wat het makkelijk maakt om deze om te zetten in objecten en uit te lezen. Verder blijkt uit benchmarking tests dat JSON lichter en sneller is dan XML[43].

4.4.1 JSON-schema structuur

Ieder JSON-schema beschikt over de volgende structuur:

- $schema; geeft aan dat een JSON-document een JSON-schema is. Hiernaast geeft de waarde aan welke schema versie dit schema voldoet[41]. Dit kan ook een aangepast schema zijn die afwijkt van het originele schema, om bijvoorbeeld, uitgebreide functionaliteit te ondersteunen.
- $id; Definieert een Uniform Resource Identifier (URI) voor het schema. Deze URI kan gebruikt worden om te refereren naar het bijbehorende schema.
- title; Titel van het schema
- description; Omschrijving van het schema; waar het schema voor staat.
- definitions; Optionele sectie waarin schema definities geplaatst kunnen worden die in het daadwerkelijke schema hergebruikt kunnen worden[44].
- Het daadwerkelijk schema waartegen JSON-documenten gevalideerd worden. Een voorbeeld is terug te zien in figuur 4.8.

4.4.2 Referenties

Om het schema schaalbaar op te zetten zullen mogelijke properties van content typen gedefinieerd worden onder 'definitions'. Een voorbeeld hiervan is terug te zien in figuur 4.10. Deze properties kunnen vervolgens worden (her)gebruikt in de daadwerkelijke content typen, door gebruik van het $ref keyword[41]. Dit keyword refereert naar een ander schema via het schema $id.

```
1  {
2      "$schema": "http://json-schema.org/draft-07/schema#",
3      "$id": "http://www.ranjnet.nl/content#",
4      "definitions": {
5          "propertyTypes": {
6              "stringProperty": {
7                  "$id": "#/definitions/propertyTypes/stringProperty",
8                  "type": "string",
9                  "default": ""
10             }
11         }
12     }
13 }
```

Figuur 4.10: Voorbeeld content type JSON-Schema.

Met deze properties kan er een base content schema worden opgezet die properties bevat die in elk content type terugkomen. In figuur 4.11 staat een voorbeeld van een simpel base content schema die gebruik maakt van de eerder gedefinieerde string property.

```
{
    "baseContent": {
        "type": "object",
        "properties": {
            "type": {
                "$ref": "#/definitions/propertyTypes/stringProperty"
            }
        }
    }
}
```

Figuur 4.11: JSON-schema van base content.

4.4.3 Combinaties

Door een 'base content' schema te definiëren met properties die voorkomen in ieder content type schema wordt de grootte van het JSON-schema minimaal gehouden. Hergebruikt van andere schema's vergroot de schaalbaarheid van het schema. Het 'base content' schema moet gecombineerd kunnen worden met de daadwerkelijke content types die gebruikt zullen worden in de editor en het NGT. Hiervoor maakt JSON-schema gebruik van het 'allOf' keyword. Een schema met het 'allOf' keyword is pas valide wanneer de onderliggende schema's, gespecificeerd in 'allOf', valide zijn[41]. Het schema in figuur 4.14 definieert 'text content', welke valide is als de twee onderliggende schema's valide zijn. De '$ref' en 'allOf' keywords brengen complicaties met zich mee die in een later hoofdstuk besproken zullen worden.

4.5 Het aanpassen van content type properties

De properties van een content node kunnen worden aangepast door de gebruiker. Het selecteren van een content node resulteert in het tonen van de bijbehorende properties. De inspector (te zien in figuur 4.13) is verantwoordelijk voor het tonen van deze properties.

In de nieuwe editor moet de inspector de properties tonen van het bijbehorende schema. Bij het opzetten van het content schema is er gebruik gemaakt van referenties om properties toe te kennen aan content typen. Dit resulteert in een schaalbaar JSON-schema, maar introduceert een probleem voor de inspector. Deze kan niet omgaan met referenties en combinaties van schema's.

4.5.1 Het content schema voorbereiden

Om het content schema bruikbaar te maken voor de inspector moeten de volgende stappen worden ondernomen:

1. Referenties resolven; '$ref' keywords vervangen door het daadwerkelijke gerefereerde object.
2. Combinaties platslaan; 'allOf' keywords vervangen door een samenvoeging van onderliggende schema's.

```
{
    "textContent": {
        "allOf": [
            {
                "$ref": "#/definitions/baseContent"
            },
            {
                "$id": "#/contentTypes/textContent",
                "title": "Text content",
                "description": "Textual content",
                "properties": {
                    "text": {
                        "$ref": "#/definitions/propertyTypes/stringProperty"
                    }
                }
            }
        ]
    }
}
```

Figuur 4.12: Een content schema die gebruik maakt van het 'allOf' keyword.

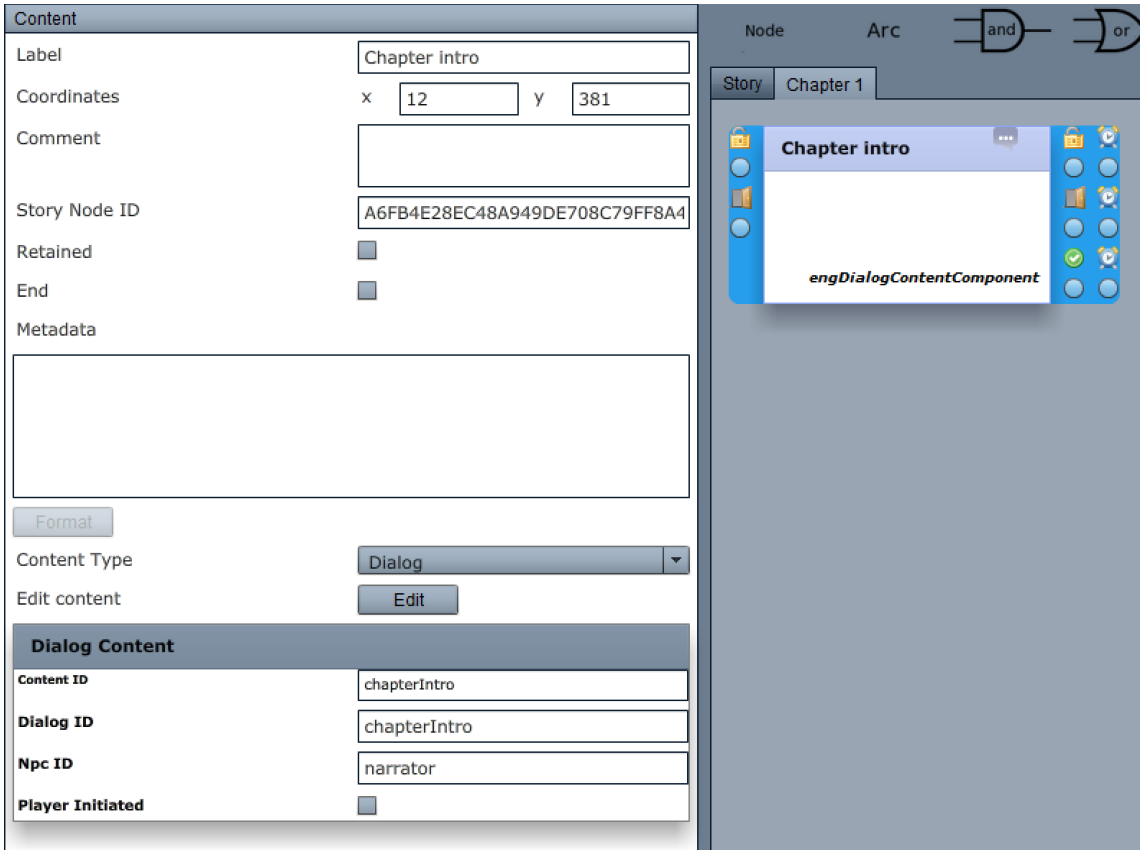
Deze stappen moeten worden uitgevoerd wanneer de editor opstart, omdat het wenselijk is om deze functionaliteit te hebben bij het opzetten van het schema. De editor maakt dan een interne copy van het schema en zet deze om naar een schema zonder referenties en combinaties.

Het oplossen van referenties

Om referenties te resolven moet er eerst gekeken worden naar hoe het $id keyword werkt. In de root, het hoogste niveau, van het JSON-schema wordt de locatie van het schema aangegeven. Het schema zelf dient toegankelijk te zijn op deze locatie om referenties vanuit andere schema's toe te laten [**?**]. Binnen content type schema's kan gebruik gemaakt worden van een hashtag (#) om te refereren naar het root schema.

Als voorbeeld voor het resolven van referenties wordt er gekeken naar figuur 4.15. Deze specificeert een schema voor 'base content' waarin gerefereerd wordt naar een ander schema, namelijk 'string property'. Het pad van de referentie luidt: "#/definitions/propertyTypes/stringProperty". Hierin verwijst de hashtag naar de root van het schema: "http://www.ranjnet.nl/content". Het bijbehorende object kan gevonden worden door vanaf de root te reduceren volgens het pad. Vervolgens wordt het $ref keyword vervangen door het gevonden schema. De community rondom JSON-schema leidt ook tot bestaande oplossingen zoals 'json-schema-ref-parser'[1] en 'json-schema-deref'[2]. Beide oplossingen zijn libraries die $ref keywords resolven.

[1] https://github.com/BigstickCarpet/json-schema-ref-parser
[2] https://github.com/bojand/json-schema-deref



Figuur 4.13: De inspector die een geselecteerde dialog node toont.

Het platslaan van combinaties

Combinaties gemaakt door het 'allOf' keyword kunnen worden platgeslagen door de onderliggende schema's samen te voegen in één schema object. Er wordt een object gemaakt waarnaar de velden van de onderliggende schema's gekopieerd worden. Dit proces begint bij het eerste onderliggende schema wat betekent latere schema's eventuele al bestaande velden zullen overschrijven. Hoewel deze stap nodig is om het schema bruikbaar te maken voor de inspector is het samenvoegen van onderliggende schema's niet juist volgens de definitie van het 'allOf' keyword. Volgens JSON-schema is een schema met 'allOf' pas valide wanneer elk onderliggend schema valide is[44]. Dit betekent dat onderliggende schema's niks van elkaar af weten. Door onderliggende schema's plat te slaan naar één schema wordt deze barrière gebroken wat leidt tot een overtreding van JSON-schema.

```
{
    "textContent": {
        "allOf": [
            {
                "$ref": "#/definitions/baseContent"
            },
            {
                "$id": "#/contentTypes/textContent",
                "title": "Text content",
                "description": "Textual content",
                "properties": {
                    "text": {
                        "$ref": "#/definitions/propertyTypes/stringProperty"
                    }
                }
            }
        ]
    }
}
```

Figuur 4.14: Een content schema die gebruik maakt van het 'allOf' keyword.

4.5.2 Reflecteren van content types in de inspector

De inspector moet het content type van een geselecteerde content node inzichtelijk en bewerkbaar maken. Bij 'text content' schema, die bestaat uit een string property, moet de inspector dan ook een tekstveld tonen waarmee de tekst in het content type kan worden aangepast.

Primitieve types

Omdat content types gedefinieerd zijn in JSON-schema's, moet de inspector alle zes primitieve types[41] ondersteunen om het schema te kunnen reflecteren. De selectie van primitieve types in JSON-schema bestaan uit:

1. string
2. boolean
3. number
4. object
5. array
6. null

Ieder type heeft een eigen representatie. Met React kan de representatie van ieder type worden ingekapseld in componenten. In figuur 4.16 wordt een voorstel gedaan op de representatie van de types[3].

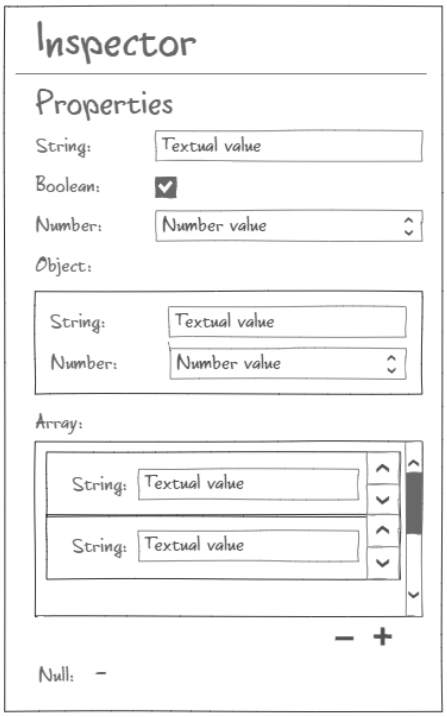
[3] "null" heeft geen representatie

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "$id": "http://www.ranjnet.nl/content#",
    "definitions": {
        "propertyTypes": {
                "stringProperty": {
                "$id": "#/definitions/propertyTypes/stringProperty",
                "type": "string",
                "default": ""
            }
        }
    },
    "baseContent": {
        "type": "object",
        "properties": {
            "type": {
                "$ref": "#/definitions/propertyTypes/stringProperty"
            }
        }
    }
}
```

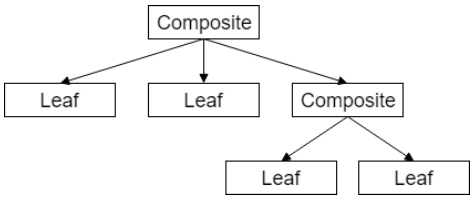
Figuur 4.15: Een JSON-Schema met referentie.

Compositie

Het voorstel in figuur 4.16 maakt gebruik van een compositie structuur; types kunnen voorkomen in andere types. Zo komen de types ‘string’ en ‘number’ terug in het object type. Een compositie structuur bestaat uit een boom van ‘composites’ en ‘leaves’. Composite objecten bevatten andere composite- en leaf objecten (figuur 4.17), zo zijn de types ‘object’ en ‘array’ composities. Leaf objecten beheren geen onderliggende objecten. De boom is valide wanneer deze begint met een compositie en eindigt met blaadjes.



Figuur 4.16: Representatie van primitieve JSON-schema types.



Figuur 4.17: Een valide compositie boom.

4.6 Conclusie

Met deze conclusie wordt de deelvraag “Hoe kan diverse game content ondersteund worden?” beantwoord.

Diverse game content kan worden ondersteund door gebruik te maken van content typen. Het specificeren van content typen door middel van dataschema’s maakt het beheren van content typen mogelijk en draagt bij aan de flexibiliteit van de editors. JSON-schema is een licht en snel dataschema formaat geschreven in JSON. Door middel van referenties en combinaties kan er een schaalbaar dataschema worden opgezet. Omdat JavaScript JSON out-of-the-box ondersteunt is deze goed te integreren met de nieuwe technology stack die gebruik maakt van JavaScript. Hiernaast zijn er bestaande oplossingen in de vorm van libraries voor het valideren, manipuleren en reflecteren van JSON-schema’s.

4.7 Vervolgonderzoek

Vervuiling van de content typen selectie Om vervuiling van de content typen selectie te voorkomen zullen de JSON-schema’s deel uit gaan moeten maken van het project. Zo beschikt iedere game over een selectie aan content typen die voor haar relevant is. In de huidige situatie staan de editors los van de game engine. Om game content te kunnen bewerken die specifiek is voor een project zullen de editors opgenomen moeten worden in een overkoepelde projectstructuur.

JSON-schema combinatie keywords Met deze oplossing kan er geen gebruik gemaakt worden van andere combinatie keywords die JSON-schema biedt. Naast het ‘allOf’ keyword zijn er ook nog de combinatie keywords: ‘anyOf’, ‘oneOf’ en ‘not’.

Hoofdstuk 5

Formalismen

Hoofdstuk 6

Conclusies

Hoofdstuk 7

Discussie

Hoofdstuk 8

Reflectie

Hoofdstuk 9

Referenties

[1] About Github.

[2] Apache Flex®.

[3] Apache Flex® - Download Apache FlexJS.

[4] Blizzard/qt: Blizzard's additions/modifications to Qt5.

[5] Compiler Targets - Haxe - The Cross-platform Toolkit.

[6] Cutelyst.

[7] Diagram Scene Example — Qt Widgets 5.11.

[8] Download Qt: Choose commercial or open source.

[9] Electron — Bouw cross-platform desktop apps met JavaScript, HTML, en CSS.

[10] Electron Apps — Electron.

[11] electron/electron: Build cross platform desktop apps with JavaScript, HTML, and CSS.

[12] FlexJS (legacy) - Apache Flex - Apache Software Foundation.

[13] Haxe - The Cross-platform Toolkit.

[14] Haxe - The Cross-platform Toolkit.

[15] Haxe Summit 2018 in Seattle.

[16] Home — Qt Forum.

[17] JSON Schema — The home of JSON Schema.

[18] Language Bindings - Qt Wiki.

[19] List of apps and companies using nw.js · nwjs/nw.js Wiki.

[20] Newest 'flex' Questions - Stack Overflow.

[21] Newest 'nwjs' Questions - Stack Overflow.

[22] Newest 'qt' Questions - Stack Overflow.

[23] NoFlo — Flow-Based Programming for JavaScript.

[24] nwjs/nw.js: Call all Node.js modules directly from DOM/WebWorker and enable a new way of writing applications with all Web technologies.

[25] Open Source & SaaS Tools — StackShare.

[26] Qt - Valve Developer Community.

[27] Qt — Cross-platform software development for embedded & desktop.

[28] Qt enhances user experience on AMD's Radeon Software Crimson Edition graphic software - Qt.

[29] Qt Interface - VideoLAN Wiki.

[30] Qt WebKit - Qt Wiki.

[31] Release v0.1.0: Update node: use node's implementation of setImmediate. · electron/electron.

[32] Stack Overflow Developer Survey 2018.

[33] Trending Haxe repositories on GitHub this month.

[34] World Wide Haxe 2014 - Haxe - The Cross-platform Toolkit.

[35] Wt, C++ Web Toolkit — Emweb.

[36] All the Major Browsers Will Soon Block Flash, is Your Website Ready?, 2016.

[37] What Is a Technology Stack and Why Do I Need One, 2017.

[38] Adobe. Flash & The Future of Interactive Content — Adobe Blog, 2017.

[39] Alfred V Aho, Jersey E John Hopcroft, and Jeffrey D Ullman. *Data Structures and Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1983.

[40] Shane Curcuru. What is Apache Flex? Website Branding Review — Community Over Code, 2016.

[41] Michael Droettboom. *Understanding JSON Schema*. 1.0 edition, 2016.

[42] Madhuri A Jadhav, Balkrishna R Sawant, Anushree Deshmukh, and Navi Mumbai. Single Page Application using AngularJS. *International Journal of Computer Science and Information Technologies*, 6, 2015.

[43] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of JSON and XML Data Interchange Formats: A Case Study. In *CAINE*, 2009.

[44] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of JSON Schema. 2016.

[45] Giuseppe Psaila. Virtual DOM: an Efficient Virtual Memory Representation for Large XML Documents. Technical report, Universita degli Studi di Bergamo, Italy, 2008.

[46] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. A Large Scale Study of Programming Languages and Code Quality in Github. Technical report, University of California, Davis, 2014.

[47] Sawicki Kevin. Atom Shell is now Electron — Electron Blog.

[48] Charles Severance. Discovering JavaScript Object Notation. *IEEE Computer Society*, 2012.

[49] Stack Overflow. Stack Overflow Developer Survey 2018, 2018.

[50] Zhao Cheng. From node-webkit to Electron 1.0.