Een flexibele tool voor het vertellen van diverse digitale interactieve verhalen

Swen Meeuwes; 0887127@hr.nl

Studentennummer: 0887127
Datum: 4 juni 2018
Versie: 0.1

Een flexibele tool voor het vertellen van diverse digitale interactieve verhalen

Instituut voor Communicatie, Media en Informatietechnologie - Informatica
Hogeschool Rotterdam
Nederland

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

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

Datum: 4 juni 2018
Versie: 0.1

Voorwoord

Niet beschikbaar in concept versie.

Samenvatting

Niet beschikbaar in concept versie.

Management Summary

Not available in concept version.

Inhoudsopgave

Hoofdstuk 1

Introductie

1.1 Achtergrond en aanleiding

Rond 2008 begon &ranj narratieven te verwerken in serious games om op verhalende wijze positieve gedragsverandering, zoals gezonder leven, toe te passen. Door gebruik te maken van de standaardisatie wist het bedrijf op een efficiënte manier meer dan 400 narrative game oplossingen te realiseren op klant specifieke problemen.

De narrative games van &ranj dienen als een simulatie; een veilige omgeving waarin vooral soft skills beoefend kunnen worden. Soft skills zijn leiderschap, probleemgerichtheid en communicatieve vaardigheden. In het werkveld is er steeds meer vraag naar deze skills[1][2][3]. Echter zijn soft skills minder makkelijk te leren en te toetsen op scholen[4]. Hiernaast kunnen dergelijke soft skills technische vaardigheden, ook wel hard skills genoemd, bevorderen[5]. Met deze games brengt &ranj lastige onderwerpen ter tafel zoals peace building[1] en discriminatie [2]. Het doel van deze simulaties is dan ook om de speler bewust te maken van de gemaakte keuzes en positieve gedragsverandering te stimuleren.

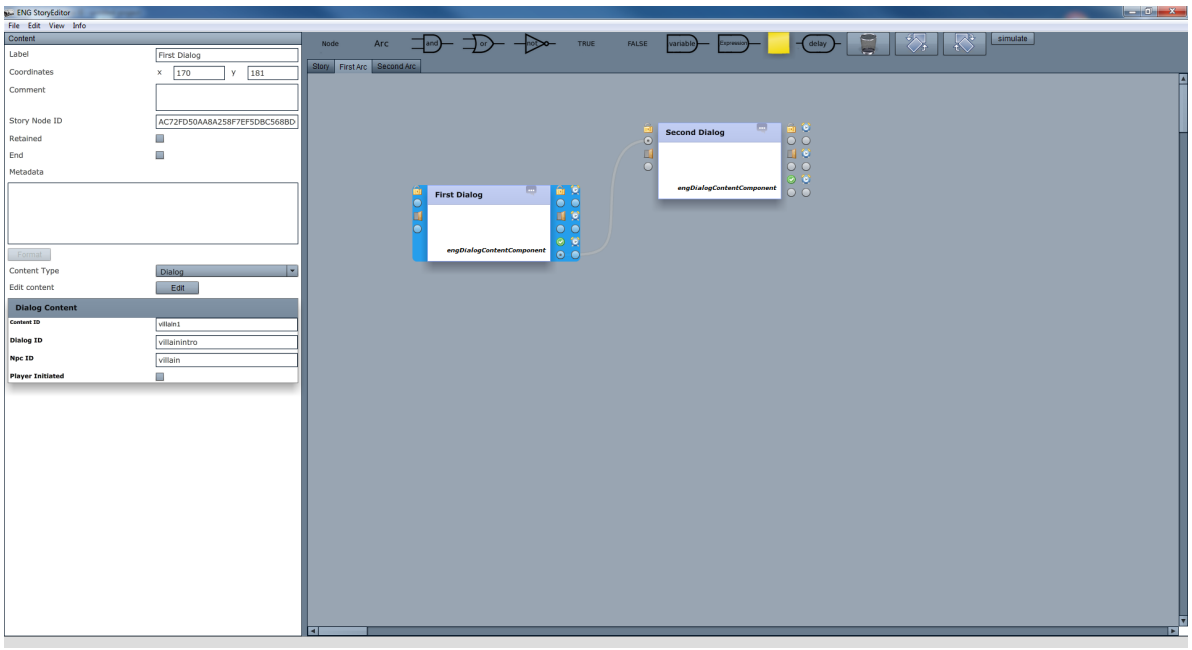
Een narrative game bestaat uit een verhaallijn waarin de speler gesprekken voert met virtuele karakters om het verhaal te vorderen. Dit is goed terug te zien in de game “FairPlay” (figuur 1.1). De games van &ranj maken gebruik van fotografie, tekst en eventuele video en voice-overs om de speler in het verhaal te plaatsen (figuur 1.1). Deze assets noemen we ook wel game content. Keuzes die de speler maakt hebben invloed op de loop en uitkomst van het verhaal. De speler krijgt directe feedback op zijn of haar gemaakte keuzes. Zo reflecteert het spel op eerder gemaakte keuzes in de vorm van een dialoog of report.

Figuur 1.1: Een dialoog in de narrative game ”Fair Play”.

Met nieuwe hedendaagse technologieën wilt &ranj haar narrative games naar het volgende niveau tillen. Deze toekomstblik noemen ze ‘narrative 2.0’ en staat op de roadmap van &ranj. Als use case voor narrative 2.0 beschrijft het bedrijf een ‘bad news’ dialoog; een gesprek waarbij de speler slecht nieuws moet brengen. Een voorbeeld van een bad news gesprek is een ‘negatieve’ diagnose die als dokter gegeven moet worden. Deze dialogen willen ze zo realistisch mogelijk maken; succes hangt af van timing en gevoel. Realisme in games wil het bedrijf bereiken door onder andere artificial intelligence (AI) oplossingen toe te passen.

Om narrative 2.0 te ondersteunen is er vraag naar tooling, externe hulpprogramma(s) die productiviteit bevorderen[6], die zal helpen bij het ontwikkelproces van de toekomstige narrative games. Het bedrijf beschikt over twee applicaties waarin narratieven geschreven kunnen worden zonder enige programmeerkennis. Deze applicaties heten de story- en dialog editor. In deze bewerkers kunnen verhalen worden gemoduleerd. Ze bestaan uit een interface waarin content types in nodes worden gerepresenteerd. Een content type is een datastructuur met een betekenis in de game. Het content type ‘ImageContent’ zou bijvoorbeeld als doel hebben

[1]https://ranj.com/projects/corporate/development#missionzhobia
[2]https://ranj.com/projects/education#fair-play



Figuur 1.2: Visual scripting in de Story Editor.

om een afbeelding tonen. Tenslotte verbinden edges de content met elkaar (figuur 1.2).

Deze story- en dialog editor heeft het bedrijf in 2008 opgezet met behulp van de Apache Flex SDK en ActionScript3. Over de jaren heen zijn de verwachtingen van de bewerkers veranderd, maar ze zijn niet uitgebreid omdat de achterliggende softwarearchitectuur niet houdbaar en flexibel is. Dit zorgt voor problemen bij het ontwikkelproces van zowel huidige als toekomstige narrative games zoals de 'bad news' game. &ranj wilt inzicht krijgen in wat er nodig is om een nieuwe editor op te zetten die oplossingen biedt op de huidige problemen.

1.2 Probleemstelling

De story- en dialog editor zijn opgezet om medewerkers zonder programmeerkennis game content zelfstandig te laten bewerken. Deze tooling wordt gebruikt in alle narrative game projecten en is essentieel voor het ontwikkelproces en productiviteit. Hiernaast bevordert het itereren over game content en maakt deze inzichtelijk door de game content te visualiseren. Hierbij speelt flexibiliteit een grote rol. De editors moeten flexibel genoeg zijn om bruikbaar te zijn voor verschillende narrative game projecten.

Het bedrijf werkt toe naar nieuwe tooling en wilt inzicht verkrijgen in het opzetten van flexibele oplossingen. Dit leidt dan ook tot de centrale onderzoeksvraag:

"Hoe kan er een flexibele tool worden opgezet voor het vertellen van diverse digitale interactieve verhalen?"

1.2.1 De technology stack

Het huidige gebruikte platform, Apache Flex, heeft weinig libraries die gebruikt kunnen worden voor de interface van de editors. Libraries zijn collecties aan code en configuraties die gebruikt kunnen worden in andere applicaties. Meestal richten ze zich op een veelvoorkomend probleem, zodat programmeurs niet telkens het wiel opnieuw uit hoeven te vinden. De libraries die beschikbaar zijn voor Apache Flex, zoals 'Flex Wires' die de verbindingen tussen nodes mogelijk maakt, hebben weinig functionaliteit en bevatten bugs. &ranj wilt naar een platform met een grotere community en meer flexibiliteit in frameworks en libraries, zodat het wiel niet telkens opnieuw uitgevonden hoeft te worden.

Hiernaast is het ultieme einddoel om een webapplicatie te maken waarin meerdere personen tegelijk aan een narratief kunnen werken. Collaborative editing, het ondersteunen van aanpassingen door meerdere personen tegelijk, kan het makkelijker maken om samen met klanten te werken.

1.2.2 Diversiteit in game content

Uit eerdere projecten die verschillen in game content blijkt dat de huidige editors niet flexibel genoeg zijn. Het probleem ligt vooral aan de diversiteit in game content tussen projecten. Echter is dit wel een eis sinds &ranj voor verschillende klanten werkt waarvan ieder een eigen beroepenveld heeft. Dit dwingt diversiteit in game content af.

In de game content speelt semantiek een grote rol. De interpretatie van een content type hangt af van haar semantische eigenschappen. In de huidige editors hebben content types een statische definitie. Om nieuwe content types toe te voegen moeten de content types gedefinieerd worden in de broncode; content types zijn hard gebakken in de editors. Het opzetten, uitbreiden, compileren en deployen van de editors is een langdurig proces dat met de hand uitgevoerd moet worden. Voor ieder project moeten er nieuwe editors met de benodigde content types worden gebouwd. Hierbij moet er ook nog eens op gelet worden dat oudere content types functioneel blijven, zodat de editors de data van oudere narrative games nog kunnen uitlezen. Dit maakt het erg lastig om de huidige story- en dialog editor in te zetten voor verschillende projecten vanwege de tijd en dus geld die het kost om content types toe te voegen. Vaak wordt dit ondanks de kosten wel gedaan wat resulteert in een uitbreidende selectie aan content types. Echter zijn deze content types meestal zo project specifiek dat ze niet in meer dan één project gebruikt worden. Dit resulteert in een onoverzichtelijke editor, omdat de niet gebruikte content types wel in de editor zitten.

1.2.3 Het ondersteunen van meerdere formalismen

In de huidige editors bestaat er een nauwe koppeling tussen de visual scripting interface en het achterliggende formalisme. Dit maakt het lastig om van formalisme te veranderen, laat staan nieuwe formalisme te ondersteunen. Deze beperking verlaagd de flexibiliteit en inzetbaarheid van de editors. In de roadmap van het bedrijf wordt beschreven hoe &ranj een 'smart follower' wilt zijn onder andere op het gebied van artificial intelligence (AI). Er gebeurt erg veel op het gebied van AI en &ranj ziet mogelijkheden om deze nieuwe technologie toe te passen in toekomstige games. Door op de hoogte te blijven van AI kan het bedrijf nieuwe aantrekkelijke oplossingen verwerken in hun games en deze verkopen aan de klant. Met de huidige story- en dialog editor is het vrijwel onmogelijk om gebruik te maken van AI. Het achterliggende formalisme biedt alleen de mogelijkheid om content types aan elkaar te verbinden en zo een graph te vormen. Om AI toe te passen in narrative games moet er gebruik gemaakt worden van formalismen die dit ondersteunen. In tegenstelling tot het huidige formalisme waarin opeenvolgende instructies gedefinieerd worden bieden behaviour trees een meer AI georiënteerde oplossing[6][7].

1.2.4 Overkoepelende projectstructuur

Hoewel de dialog- en story editor het narratief ordent en aan elkaar verbindt beheert zij de bijbehorende game content niet; er is geen sprake van een overkoepelde projectstructuur. Een voorbeeld hiervan is hoe de huidige editors om gaan assets, bestanden die gebruikt worden in de game. Er moet handmatig een manifest worden bijgehouden waarin assets een sleutel (id) en semantiek toegekend krijgen (zie figuur 1.3). Deze sleutel wordt gebruikt in de huidige editors om te kunnen refereren naar assets, zoals geluid, afbeeldingen en videos. Zo refereert het content type 'image content' naar een asset via een sleutel. Deze asset zou een afbeelding moeten zijn, maar er bestaat geen zekerheid over dat de asset een afbeelding is of überhaubt bestaat. Dit zorgt voor een zwakke link tussen de editors en game content, wat het erg gevoelig maakt voor (menselijke)fouten. Zo hoeft er maar één typefout gemaakt of het pad aangepast te worden om de referentie naar de game content te verliezen.

```
{
    "scenario_test": [{
        "id": "story_scenario_test",
        "type": "json",
        "src": "assets/scenarios/scenario_test/story.json"
    }],
    "sounds": [{
        "id": "backgroundmusic",
        "type": "sound",
        "src": "assets/sounds/backgroundmusic.mp3"
    }]
}
```

Figuur 1.3: Game content manifest.

1.3 Doelstelling

1.3.1 Vanuit &ranj

Het bedrijf hoopt in 2019 nieuwe editor(s) op te zetten die het ontwikkelproces van narrative games zullen ondersteunen. Om de diversiteit in klanten en de toekomstplannen van de editors te ondersteunen moeten de nieuwe editor(s) flexibel opgezet zijn, zodat het op maat maken van de games mogelijk is. Zo moet de nieuwe editor kunnen omgaan met de diversiteit in game content, een deel uit maken van een overkoepelende projectstructuur en het achterliggende formalisme scheiden van de user interface.

1.3.2 Uit persoonlijk interesse

Een zelfstandig eigen project drie jaar terug was mijn eerst aanraking met het concept van een narrative game. Er was een duidelijke behoefte naar tooling om het verhaal in te schrijven. Het makkelijk kunnen aanpassen van het verhaal was een probleem, omdat er geen externe tooling voor bestond die goed aansloot bij de wensen van het project destijds. Hoewel het een erg leuk project was om naast mijn studie op te pakken, bracht het veel werk met zich mee waardoor het nooit gelukt is om het project af te ronden. Dit onderzoek kan inzicht bieden in hoe dit beter aangepakt had kunnen worden.

1.3.3 Vanuit school

Het onderzoek moet gedocumenteerd worden in de vorm van een scriptie. Deze scriptie moet voldoen aan de beoordelingscriteria die vooraf opgesteld zijn. Er moet tijdens het onderzoek voldaan worden aan 4 competenties: beheren, analyseren, adviseren en ontwerpen. Tenslotte moet er sprake zijn van een professionele houding en adequate communicatie met de stakeholders. De beoordeling volgt uit de scriptie en eindpresentatie.

1.4 Onderzoeksvragen

1.4.1 Hoofdvraag

Vanuit het probleem is de volgende centrale onderzoeksvraag opgesteld:

"Hoe kan er een flexibele tool worden opgezet voor het vertellen van diverse digitale interactieve verhalen?"

1.4.2 Deelvragen

Om de centrale onderzoeksvraag te kunnen beantwoorden zijn de volgende deelvragen opgesteld:

- Hoe kan er een 'technology stack' opgezet worden die een flexibele basis biedt voor een editor waarmee diverse digitale interactieve verhalen verteld kunnen worden?
- Hoe kan diverse game content ondersteund worden?
- Hoe kan de architectuur achter de editor zo worden opgezet dat er in latere stadia nieuwe narratieve formalismen makkelijk doorgevoerd kunnen worden?
- Hoe kan game content in de overkoepelende projectstructuur verbonden en geordend worden?

1.5 Scope

Het onderzoek vindt plaats over een tijdsduur van 20 weken. Omdat er gewerkt wordt met een tijdslimiet is het belangrijk om een scope vast te leggen. Naast dat deze afbakening overzicht en focus biedt voorkomt deze misverstanden tussen de betrokken partijen.
Dit onderzoek omvangt:

- Het doorspitten van de broncode achter de huidige editors.
- Eventuele prototypes om standpunten te onderbouwen.
- Advies op het gebied van ontwikkelplatformen, libraries en architectuur.
- Het in kaart brengen van de huidig gebruikte technologieën.
- Een scriptie schrijven met daarin adviezen en conclusies over het desbetreffende onderwerp.

Er zal geen aandacht besteed worden aan:

- Het ontwikkelen van een productie klare editor die direct inzetbaar is voor toekomstige narrative game projecten.
- Een narrative game ontwikkelen met de eventuele gemaakte prototypes.
- Verdieping in klant co-creatie.
- Onderzoek naar ‘collaborative editing’.
- De deployment pipeline van de editors.
- Optimalisatie van de editors.
- Het dataformaat waarmee de editors communiceren met de game engine.

1.6 Structuur

Deze paragraaf geeft inzicht in de komende hoofdstukken van de scriptie en dient als een kapstok voor de lezer. Zo wordt er bij ieder hoofdstuk beschreven wat de lezer kan verwachten.

Hoofdstuk 2 omschrijft welke methodes er gebruikt zijn om tot de behaalde resultaten te komen.

Hoofdstuk 3 geeft context aan het onderzoek en plaats deze in een groter plaatje.

Hoofdstuk 4 betreft de onderliggende bouwblokken waarop de huidige applicatie gebouwd is en waarop de nieuwe applicatie gebouwd zal worden; de 'technology stack'. Het belang van een goedpassende tech stack komt ter sprake en er worden adviezen gedaan over benodigde veranderingen binnen de stack die aansluiten op het toekomstbeeld bij de editors.

Hoofdstuk 5 beschrijft hoe er onderscheid wordt gemaakt tussen game content. Hiernaast gaat dit hoofdstuk in op de diversiteit in game content en hoe deze ondersteund kan worden.

Hoofdstuk 6 betreft de rol van formalisme binnen interactive story telling, de formalismen die gebruikt worden in de huidige editors en het leggen van een scheiding tussen de editors en achterliggende formalisme.

Hoofdstuk 7 gaat in op de behoefte aan een overkoepelende projectstructuur en eventuele stappen die gemaakt kunnen worden.

Hoofdstuk 8 beschrijft de eindconclusie van het onderzoek.

Hoofdstuk 9 geeft inzicht in de adviezen die voort zijn gekomen uit het onderzoek. Hiernaast komen ook eventuele aandachtspunten of vervolgonderzoek ter sprake.

Hoofdstuk 10 omvat een reflectie op de afstudeerstage het onderzoek.

Hoofdstuk 11 bevat een lijst met begrippen en verwijzingen naar waar het begrip staat uitgelegd.

Hoofdstuk 2

Bedrijf

Dit hoofdstuk biedt informatie over het bedrijf waarbij het onderzoek is uitgevoerd. Hiernaast geeft het inzicht in welke afdeling van het bedrijf het onderzoek gedaan is.

2.1 &ranj

&ranj is een internationaal bedrijf dat sinds 1999 serious game oplossingen realiseert. Dit zijn spellen waarbij de speler spelenderwijs getraind wordt. Een voorbeeld hiervan is de serious game 'Mission Zhobia – Winning the Peace'[1]. In deze game wordt de speler getraind in peace building. Het bedrijf heeft te maken met klanten in diverse vakgebieden. Hierdoor is het domein van de onderwerpen erg breed.

&ranj wilt mensen helpen ontwikkelen op een effectieve en efficiënte manier. Met behulp van 'serious games' wilt het bedrijf voor de gebruiker een ervaring creëren waarin hij of zij kennis opdoet om in het echte leven uitdagingen aan te gaan en leert problemen te tackelen. Volgens &ranj is de beste manier van leren het leren door te doen.

2.2 Visie

Met serious games past &ranj op spelenderwijze gedragsverandering toe, om zo een betere toekomst te bouwen. De visie van &ranj luidt:

"Together we build a brighter future"

2.3 Afdelingen

Het bedrijf specialiseert zich in drie verschillende richtingen met ieder zijn eigen specialiteit. Deze richtingen kunnen omschreven worden met verschillende keywords.

- Gamification; gefocust op: "consultancy", "leuk maken van minder leuke taken/ werk", "fysiek".
- Playground; gefocust op: "experimentatie" en "ervaringen".
- Corporate Learning; gefocust op: "soft skills", "volwassen organisaties", "bewezen oplossingen" en "efficiëntie".

Dit onderzoek vindt plaats in de 'corporate learning' afdeling.

[1] `https://www.missionzhobia.org/`

2.4 Corporate learning

De corporate learning afdeling van &ranj realiseert oplossingen voor “volwassen” bedrijven die soft skills bij hun werknemers willen bevorderen. Bij deze afdeling ligt de nadruk op efficiëntie en het gebruikt van oplossingen waarvan het bedrijf weet dat ze werken. Meestal zijn deze oplossingen narrative games, spellen waarin op verhalende wijze kennis wordt overgebracht.

Het ontwikkelen van een narrative game is een grote compliceert proces. Dit onderzoek zal zich alleen focussen op de ontwikkelingsfase, omdat de editors een grote rol in spelen. In deze fase werkt het team samen om het product te realiseren.

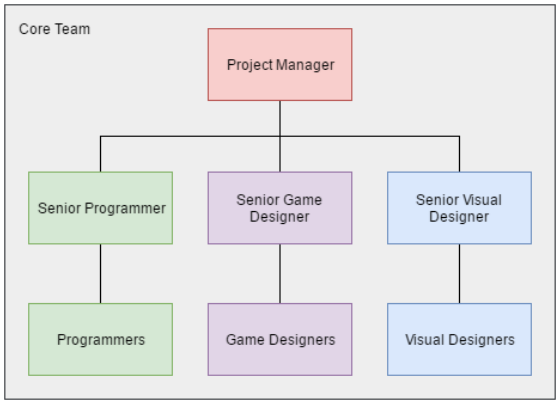
Een corporate learning development team bestaat uit de volgende disciplines:

Programmeur Is verantwoordelijk voor de technische implementatie van de game. Hiernaast biedt deze ondersteuning bij technische vraagstukken.

Game designer Is verantwoordelijk voor de mechanieken en het verhaal achter de game. Hij of zij probeert gedragsverandering toe te passen met de game als tool. Verder werken game designers nauw samen met de projectmanager om te zorgen dat de game aansluit bij de wensen van de klant.

Visual designer Is verantwoordelijk voor de visuals binnen de game. Een visual designer wilt door middel van deze visuals meestal een gevoel bij de speler ontvlammen.

Projectmanager Begeleid het ontwikkelproces en is regelmatig in contact met de klant. Hij of zij zorgt dat het project binnen het budget blijft en onderhandeld met de klant wanneer nodig. De projectmanager trekt aan de bel als het project niet aansluit bij de wensen van de klant en heeft veto op de keuzes binnen het project.



Figuur 2.1: Anatomie van een core team bij &ranj.

Het kan zijn dat er meerdere programmeurs, game designers of visual designers aan een project werken. Als dit het geval is wordt er een senior als lead aangewezen. De lead begeleid de teamleden met dezelfde rol.

2.4.1 Ontwikkelproces van narrative games

Het ontwikkelen van een narrative game kan een ingewikkeld proces zijn dat veel tijd kan kosten. Dit ligt aan de complexiteit van het verhaal en eventuele achterliggende modellen. In het verhaal kan de speler keuzes maken waarbij elke keuze meestal tot een ander pad in het verhaal leidt.

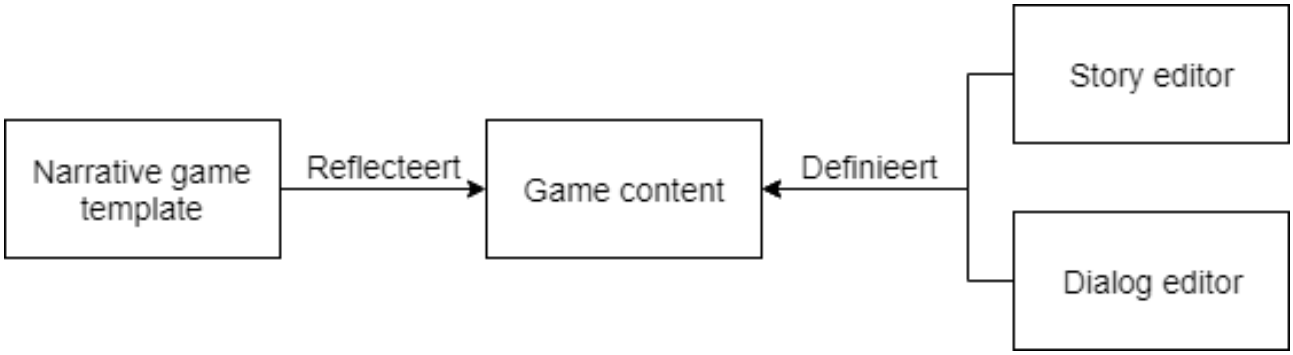
Dit maakt het onmogelijk om het verhaal te definiëren in code of een bestaand dataformaat. Tenslotte wordt het handmatig vast leggen van het verhaal in een dataformaat zoals JSON of XML al gauw onoverzichtelijk. Dit zorgt voor een probleem in het ontwikkelproces.

Om dit probleem te tackelen en het ontwikkelproces efficiënter te laten verlopen heeft het bedrijf in 2008 het ontwikkelproces van narrative games gestandaardiseerd. Onder efficiënt verstaan we in dit geval het versnellen van het proces en het voorkomen van fouten in de game.

Ter ondersteuning en standaardisatie van het ontwikkelproces zijn de story- en dialog editor gebouwd. Deze editors laten game designers zonder enige programmeerkennis verhalen en dialogen schrijven.

Om de efficiëntie van het ontwikkelproces van een narrative game verder te verhogen heeft het bedrijf een template opgezet voor het ontwikkelen van narrative games. Dit template noemen ze het narrative game

template (NGT) en dient als een geteste basis voor narrative games. Het NGT verwerkt geëxporteerde data uit de story- en dialog editor en is het deel dat uiteindelijk de verschillende content types toont/ evalueert. Een overzicht van de relaties tussen de editors en het NGT is terug te zien in figuur 2.2.



Figuur 2.2: Ontwikkelomgeving voor interactieve narratieve.

Hoofdstuk 3

Methodiek

Dit hoofdstuk omschrijft de methodes en werkwijzen die toegepast zijn tijdens dit onderzoek om tot de behaalde resultaten te komen. Deze methodes en werkwijzen worden vervolgens gekoppeld aan toekomstige hoofdstukken en geven inzicht in hoe er te werk is gegaan.

3.1 Onderzoeksmethodieken

Tijdens het onderzoek is er gebruik gemaakt van verschillende onderzoekmethodieken om zowel kennis te vergaren als te valideren. In deze paragraaf worden de gebruikte onderzoeksmethodieken gelinkt aan de deelvragen waarvoor ze gebruikt zijn.

3.1.1 Literatuuronderzoek

Tijdens dit onderzoek is er gebruik gemaakt van literatuuronderzoek om informatie over het desbetreffende onderwerp te vergaren en eventuele oplossingen te valideren. Er is gebruik gemaakt van boeken, papers en wanneer deze schaars waren artikelen, blogs en fora. Hierbij is er goed gekeken naar de betrouwbaarheid van de bron. Bronnen zijn geëvalueerd op hun betrouwbaarheid door andere bronnen met hetzelfde of een overlappend onderwerp te vergelijken. Hiernaast is er gekeken naar objectiviteit van de schrijver, de datum van de bron en welke referenties gebruikt worden om argumenten te onderbouwen.

Hoewel dit onderzoek vrij niché is en er weinig bronnen bestaan over het onderwerp, wordt er literatuuronderzoek gebruikt waar dit nuttig en mogelijk is. Bij deelvragen 1, 2, 3 en 4 wordt er gebruik gemaakt van literatuuronderzoek.

- De eerste deelvraag, hoofdstuk 4, maakt gebruik van literatuuronderzoek om de keuze achter de nieuwe technology stack te onderbouwen.
- De tweede deelvraag, hoofdstuk 5, maakt gebruik van literatuuronderzoek om kennis te vergaren over dataschema's.
- De derde deelvraag, hoofdstuk 6, maakt gebruik van literatuuronderzoek om design patterns en hun consequenties in kaart te brengen.
- De vierde deelvraag, hoofdstuk 7, maakt gebruik van literatuuronderzoek om te kijken hoe andere applicaties met een overkoepelende projectstructuur omgaan met bestanden.

3.1.2 Implementatie gedreven onderzoek

Om eventuele oplossingen op architecturele vraagstukken te valideren is er gebruik gemaakt van implementatie gedreven onderzoek. Naast validatie geeft deze methode inzicht in eventuele consequenties en andere vraagstukken die opkomen. Dit is een van de sterkere vormen van validatie, omdat de oplossing gelijk in werking wordt gezet.

Aan deze onderzoeksmethode zitten wel 2 risico's verbonden waarmee rekening gehouden moet worden[8]. Het vraagstuk moet (1) eerst duidelijk in kaart gebracht en vastgesteld worden. Als het vraagstuk veranderd, door eventuele nieuwe inzichten, is de validatie gehaald uit de implementatie niet meer geldig. Hiernaast is het (2) lastig om een generieke implementatie te gebruiker ter validatie, omdat de implementatie voor ieder system mogelijk zal verschillen.

Uit het 'implementatie gedreven onderzoek' zal een prototype komen die dient als ondersteuning en validatie van de onderzoeksresultaten en voorstellen. Dit prototype beschikt alleen over functionaliteiten waarvoor gebruik gemaakt is van deze onderzoeksmethode. Dit betekent dat het prototype **niet** dient als een inzetbaar prototype, minimal viable product of basis voor het toekomstige product; het prototype is gebouwd op "weggooi code".

Implementatie gedreven onderzoek is toegepast om deelvragen 1, 2 en 3 te valideren. Verder bracht deze methode vraagstukken naar boven die besproken zullen worden in de desbetreffende hoofdstukken.

- De eerste deelvraag, hoofdstuk 4, maakt gebruik van implementatie gedreven onderzoek om het voorstel voor de nieuwe technology stack te valideren. Hierbij zullen de nieuwe technologieën gebruikt worden in het prototype en zo gevalideerd worden.
- De tweede deelvraag, hoofdstuk 5, maakt gebruik van implementatie gedreven onderzoek om de toepassing van JSON-schema's te valideren. Het prototype zal gebruik maken van JSON-schema om diverse game content te ondersteunen.
- De derde deelvraag, hoofdstuk 6, maakt gebruik van implementatie gedreven onderzoek om de architecturele keuze te valideren. Deze keuze betreft het omzetten van de visuele structuur naar een exportbestand, waarbij formalisme gescheiden wordt van de visuele representatie.

3.1.3 Bureauonderzoek

Tijdens dit onderzoek is er gebruik gemaakt van bureauonderzoek om informatie te vergaren over populariteit van bepaalde softwareoplossingen/ technologieën. Hiernaast wordt hiermee de grootte en activiteit van community's rondom technologieën ingeschat. Dit is belangrijk voor de 'flexibele schil' die &ranj zoekt.

Bureauonderzoek wordt toegepast om informatie te vergaren in deelvragen 1 en 2.

- De eerste deelvraag, hoofdstuk 4, maakt gebruik van bureauonderzoek om de populariteit en grootte van de community rondom ontwikkelplatformen in te schatten. Hiernaast wordt hetzelfde gedaan voor diagramming libraries en user interface libraries & frameworks.
- De tweede deelvraag, hoofdstuk 5, maakt gebruik van bureauonderzoek om de community rondom JSON-schema's in te schatten. Er wordt gekeken of er bestaande oplossingen zijn betreft het manipuleren, valideren en reflecteren van JSON-schema's die gebruikt kunnen worden voor de nieuwe editor.

3.2 Werkwijzen

Tijdens dit onderzoek zijn er werkwijzen gedefinieerd die bijdragen aan het gewenste eindresultaat. In deze paragraaf wordt besproken hoe de verschillende werkwijzen toegepast zijn.

3.2.1 Wekelijkse meetings

Tijdens de stageperiode zijn er wekelijks meetings in gepland met de bedrijfsbegeleider die ook als opdrachtgever fungeerde. Deze meetings duurde rond een uur waarin de bedrijfsbegeleider op de hoogte werd gebracht. Hiernaast waren dit de momenten om een discussie aan te gaan over vraagstukken die op dat moment speelden.

Aan het begin van het onderzoek zijn deze meetings gebruikt om huidige problemen en vraagstukken in kaart te brengen. Hiernaast werd er een toekomstbeeld van de editors geschetst. Hieruit zijn de deelvragen geformuleerd en vervolgens gevalideerd met de bedrijfsbegeleider. Dit was noodzakelijk voor de scope en afbakening van het onderzoek.

Tijdens het 'implementatie gedreven onderzoek' diende de meetings als validatie. Hierbij werd de opgestelde hypothese onderbouwd met een prototype. Hiernaast werden bevindingen besproken en gevalideerd.

3.2.2 Het doorspitten van broncode

De vraag naar dit onderzoek kwam voort uit het gebruiken van de huidige editors. Om een beter beeld te krijgen van wat het probleem echt inhoud is er gekeken naar de broncode van de editors. Zo kon de huidige technology stack en de problemen die deze met zich meebracht in kaart worden gebracht. Ook werd het duidelijk waarom de grote diversiteit aan game content een probleem is in de huidige editor. Tenslotte kon er inzicht vergaard worden in verschillende formalisme achter de editors en hun nauwe koppeling met de visual scripting interface.

3.2.3 Meewerken aan een narrative project

Voor het afstudeertraject is er een jaar lang meegewerkt aan narrative game projecten. Tijdens deze projecten is er veel ervaring opgedaan met de story- en dialog editor en hun achterliggende concepten. Deze ervaring is erg waardevol voor dit onderzoek, omdat het inzicht gaf in de huidige stand van zaken. Hiernaast biedt het oplossen van veel voorkomende problemen met deze editors extra motivatie om het onderzoek af te nemen en te voltooien.

Naast deze positieve kant introduceert het ook een groot risico. Tijdens het onderzoek moest er goed op worden gelet dat er geen aannames gedaan werden en dat er niet vanuit eigen ervaring gekeken werd. Om dit tegen te gaan zijn er wekelijkse meetings ingepland met de bedrijfsbegeleider en opdrachtgever. Deze heeft een goed beeld van de huidige situatie en problemen die hierin spelen. Hij werkt samen met andere programmeurs en game designers die gebruik maken van de editors en bespreekt voorkomende problemen met de gebruikers.

3.2.4 Versiebeheer

Tijdens het onderzoek is er gebruik gemaakt van versiebeheer. Door gebruik te maken van GitHub waren de scriptie en het prototype overal en voor iedereen beschikbaar. Hierdoor kon het onderzoek uitgevoerd worden op meerdere computers en was het prototype makkelijk in te zien. Hiernaast biedt GitHub meer zekerheid dan een individuele computer op het gebied van data persistentie; als de computer crasht en alle data verloren gaat is het onderzoek niet verloren.

Voor het prototype was het van belang om eventueel terug te kunnen rollen naar een eerder werkende versie. Hiernaast kon er altijd een aftakking worden gemaakt waarin een feature geïmplementeerd kon worden zonder de werkende versie te breken.

Zowel de scriptie[1] als het prototype[2] zijn beschikbaar op GitHub.

[1] https://github.com/swenmeeuwes/thesis
[2] https://github.com/swenmeeuwes/concept-narrative-editor-framework

Hoofdstuk 4

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.

4.1 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[9].

4.1.1 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.

4.2 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 kunnen worden ingezien, deelt deze op in de volgende lagen[10]:

- **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.

[1] https://stackshare.io/

- **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

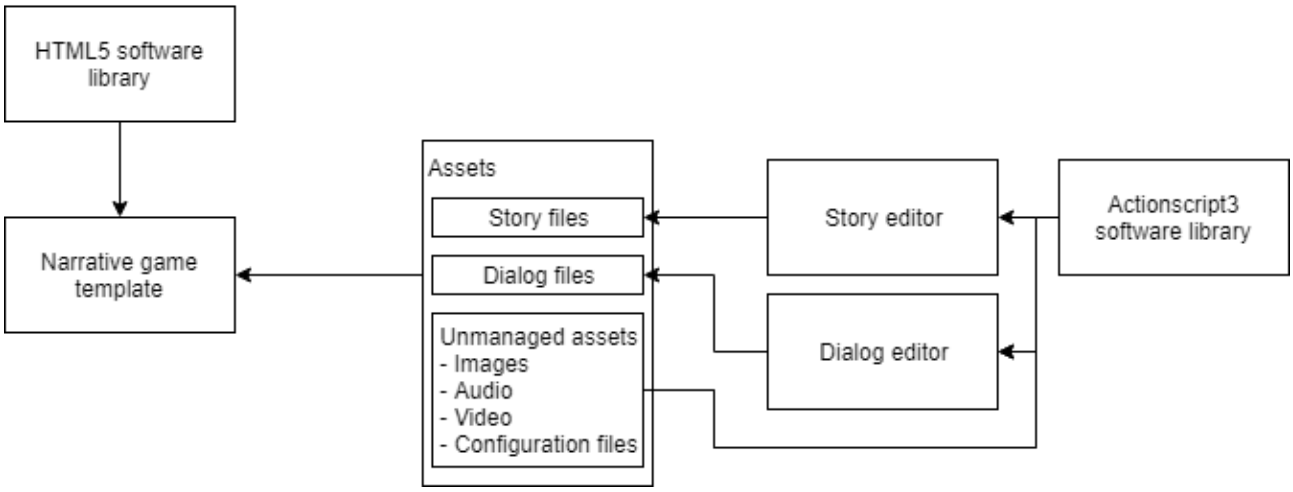
Tabel 4.1: Huidige technology stack

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 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 4.1.

4.3 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

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



Figuur 4.1: Story- en dialog editor relaties

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.

4.4 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.

4.4.1 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[11]. 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[12]. Hiernaast lijken er ook weinig updates plaats te vinden. Volgens de website van Apache Flex was de laatste update op 22 november 2017[13]. Tenslotte werd Flash al eerder in meerdere populaire browsers standaard geblokkeerd vanwege veiligheidsredenen[14]. 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.

4.4.2 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[15]. Ondanks dat de Apache Flex community wel op Stack Overflow zit[16]. 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.

4.4.3 Overtollige libraries

Zowel de editors als het NGT maken gebruik van de &ranj software library welke generieke functionaliteiten en datastructuren bevat (zie figuur 4.1). 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.

4.5 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.

4.5.1 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?

4.5.2 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[17]. 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.”[18]. 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[19]. 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[20]. 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[21]. Deze bijeenkomsten, ook wel ‘summits’ genoemd worden gehouden sinds 2014[22], wat duidt op een vraag naar het ontwikkelplatform. Hoewel de community aardig wat oplossingen en raamwerken heeft gecreëerd voor Haxe[23] 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[24].

Electron

Wat begon op 15 juli 2013[25] 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 Electron[26]. 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] [27] 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[28]. 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[28]. 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[29]. 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[30]. 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 4.3.

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

[4] https://atom.io/
[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/

Het GitHub project van NW.js heeft op het moment[10] 1.812 volgers, 33.689 favorieten en 3.745 forks[31]. Om de community samen de brengen heeft NW.js 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[32]. 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[33]. 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 4.3.

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[34]. Qt is geschreven in C++, maar ontwikkelaars kunnen ook gebruik maken van andere programmeertalen[35]. Echter wordt er aangeraden om in C++ te ontwikkelen.

Community De Qt community is actief op StackOverflow[36] en het Qt forum[37]. 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[38]. 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[39]. Verder maken bekende bedrijven zoals Valve[40], Blizzard[41], VideoLan[42] en AMD[43] 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[44][45][46]. 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 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

[10] Gemeten op: 11-05-2018 17:09
[11] https://gitter.im/nwjs/nw.js
[12] https://www.intel.com/
[13] https://github.com/nwjs/nw.js/graphs/contributors

GitHub (zie tabel 4.2). 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 4.2: NW vs Electron (Gemeten op: 11-05-2018 17:09)

Zowel NW als Electron kunnen worden beschouwd als battle tested[33][47]. 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[48].

4.5.3 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 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.

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

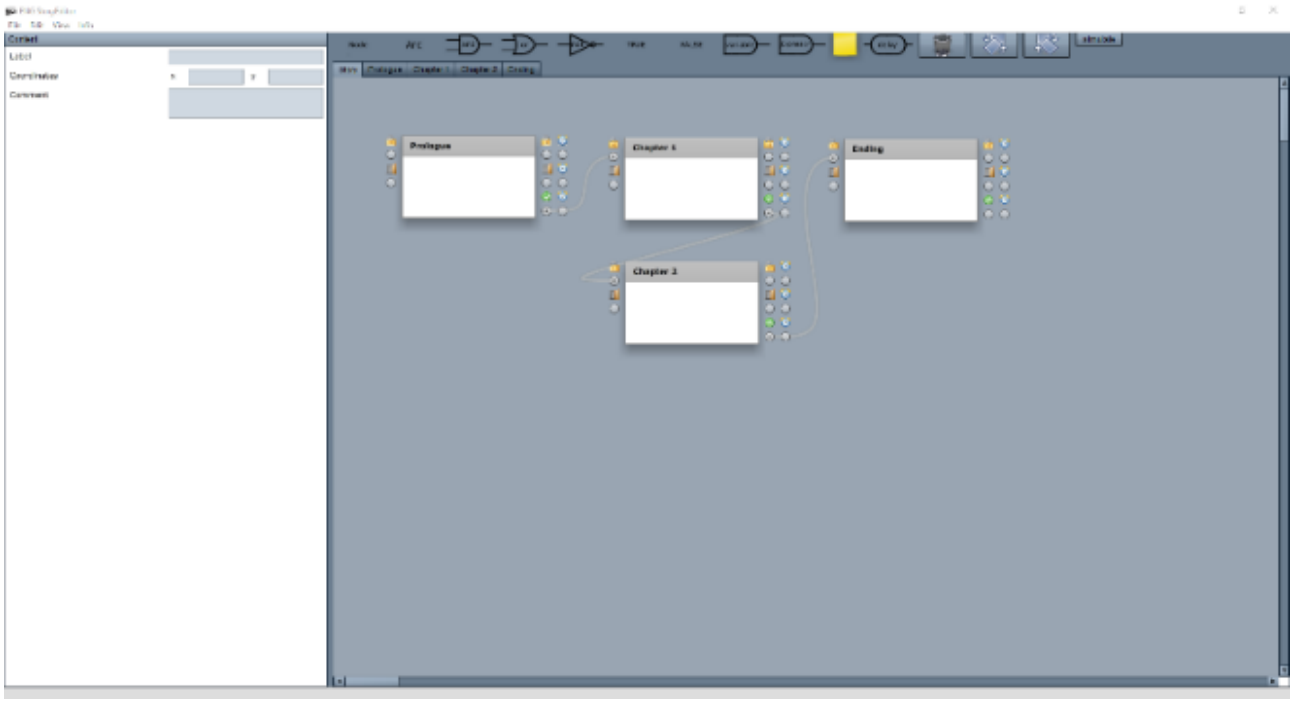
4.6 Opzetten van de user interface

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.

4.6.1 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:

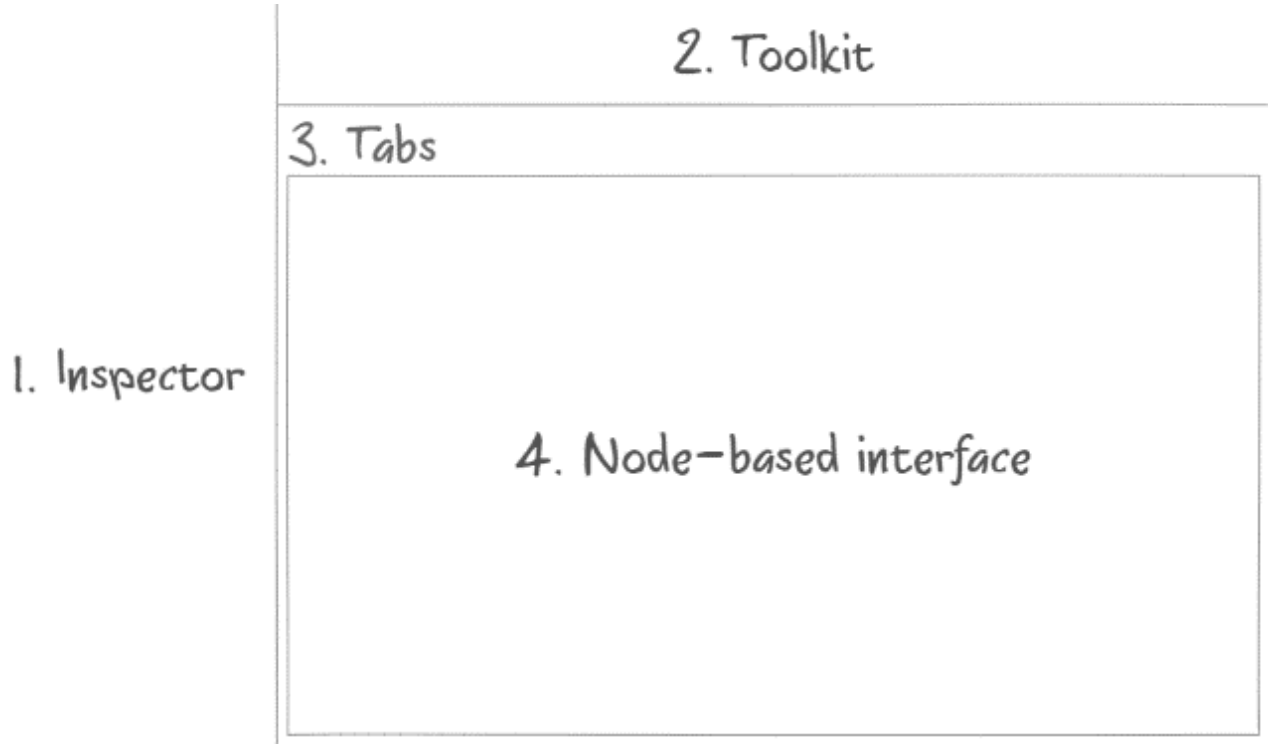
1. Inspector
2. Toolkit
3. Tabs
   (a) Canvas



Figuur 4.2: Story Editor interface

4.6.2 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.



Figuur 4.3: Editor componenten

Aandachtspunten

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

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

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

Resultaten

De geselecteerde oplossingen zijn geanalyseerd op de eerdergenoemde aandachtspunten. Het resultaat wordt weergeven in tabel 4.3.

	Vue[20]	**Angular**[21]	**React**[22]	**Ember**[23]
GitHub stats (05/03/2018)				
Volgers	4.561	2.918	5.566	1.040
Favorieten	85.500	33.694	89.726	18.682
Forks	12.545	8.275	16.971	3.850
Gebacked door	1 persoon, Yuxi (Evan) You	Google	Facebook	LinkedIn, Netflix
"Native"language	JavaScript	Typescript[24]	JavaScript	JavaScript
TypeScript ondersteuning	Ja, er zijn typing beschikbaar	Ja, out of the box	Ja, er zijn typing beschikbaar + typescript CLI[25]	Ja, er zijn typing beschikbaar + typescript CLI
Ecosysteem	Modulair, o.a. router, state management, cli, RxJS integration	Modulair, o.a. router, state management, cli, RxJS integration	Modulair. Ecosysteem opgebouwd door community (e.g. community router)	Modulair, o.a. router, state management, cli
Leercurve	Makkelijk, door de flexibiliteit die Vue biedt	Lastiger door de grote eigenzinnigheid dat het framework kent	Makkelijk, het is een relatief kleine library	Lastiger door de grote eigenzinnigheid dat het framework kent
Snelheid[26]	Snel, Virtual DOM	Snel	Snel, Virtual DOM	Langzaam
Eigenzinnigheid	Klein	Groot, je moet in de kaders van angular werken	Klein	Groot

Tabel 4.3: Inventarisatie van user interface frameworks en libraries

[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

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[49]. 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[50]. 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[51]. 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[51]. 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.

4.6.3 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 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.

[27] https://flow.org/

4.7 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[52]. 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.

4.7.1 Diagramming libraries

Visual scripting wordt toegepast op verschillende platformen[53][54][55]. 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.

4.7.2 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]
- 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 figuur 4.4.

[28] https://github.com/ariutta/svg-pan-zoom
[29] https://github.com/Ni55aN/d3-node-editor
[30] https://github.com/WebCabin/wcPlay
[31] https://github.com/clientIO/joint
[32] https://www.jointjs.com/
[33] https://github.com/jgraph/mxgraph
[34] https://gojs.net/latest/index.html

		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 4.4: Diagramming libraries en hun functionaliteiten [35]

4.7.3 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.

[35] Voor het laatst ingezien op: 16/05/2018

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.

4.8 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:

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 4.4: Huidige technology stack

Hoofdstuk 5

Diversiteit in game content

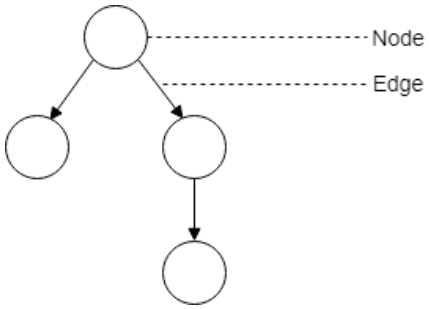
De corporate learning afdeling van &ranj houdt zich vooral bezig met het maken van narrative games op maat. Het bedrijf heeft een ontwikkelomgeving opgezet waarin narrative games efficiënter ontwikkelt kunnen worden. In dit hoofdstuk wordt er nader in gegaan op de story- en dialog editor. Dit zijn applicaties 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.

5.1 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 verwerken 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.

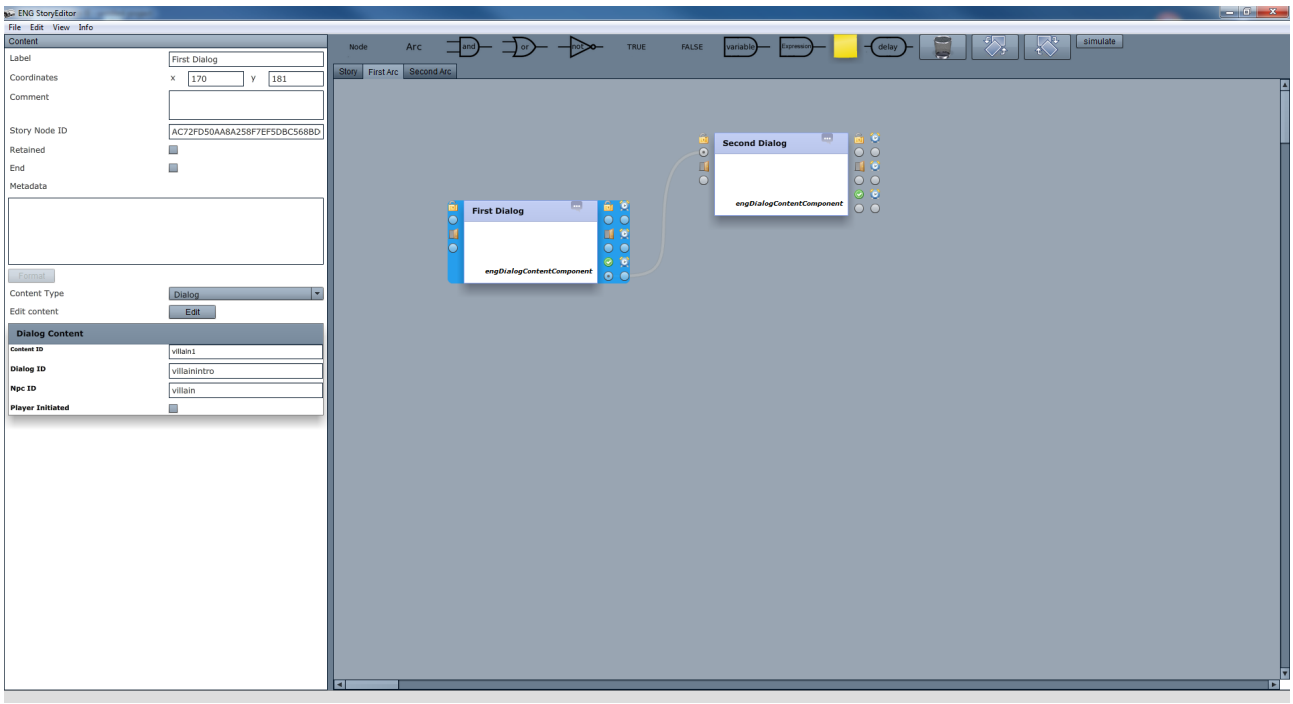
Game designers maken gebruik van de visual scripting interface die de editors bieden. Deze visual scripting interface bestaat uit visuele componenten die dienen als bouwblokken waaruit het verhaal is opgebouwd. Door deze bouwblokken op het canvas te plaatsen en met elkaar te verbinden kan er een verhaal 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 paragraaf 5.2 zal er dieper worden ingegaan op content typen.



Figuur 5.1: Visuele representatie van een graaf.

In de editors wordt een verhaal gerepresenteerd als een graph, in het Nederlands graaf genoemd[56]. Een graaf bestaat uit nodes en edges, zoals weergeven in figuur 5.1. 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 5.2).

In hoofdstuk 6 wordt er dieper ingegaan op het interpreteren van deze verhaal graaf.



Figuur 5.2: Een simpel verhaal gemaakt in de story editor

5.2 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 5.2 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.

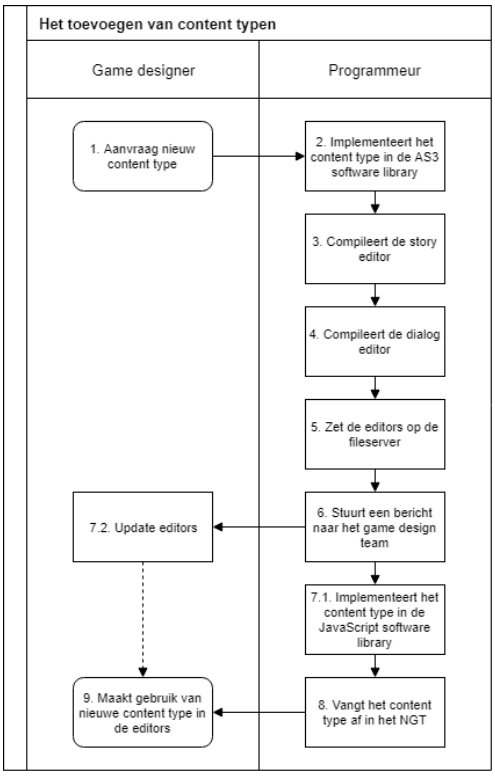
Echter hebben de meeste content typen een vrij impliciet doel. Zo kan 'text content' een tekst bevatten die uitgesproken wordt door een karakter in het verhaal, maar het kan ook een mogelijk antwoord zijn dat de speler kan geven op een vraag. Om duidelijk onderscheid te maken is het belangrijk dat content 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.

5.2.1 Vervuiling van de scope

Een nadeel van content typen met een expliciet doel is dat ze meestal maar bruikbaar zijn voor een enkel project; de herbruikbaarheid van content typen is erg laag. Vooral in projecten waarin game content erg van elkaar verschild is dit een probleem. 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 projecten die deze content niet benutten.

5.2.2 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 vormen; 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 5.3.



Figuur 5.3: 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. De game designer 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. Hierna compileert de programmeur 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 ’narrative game template’ (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 deze nog werkt. 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.

5.2.3 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.

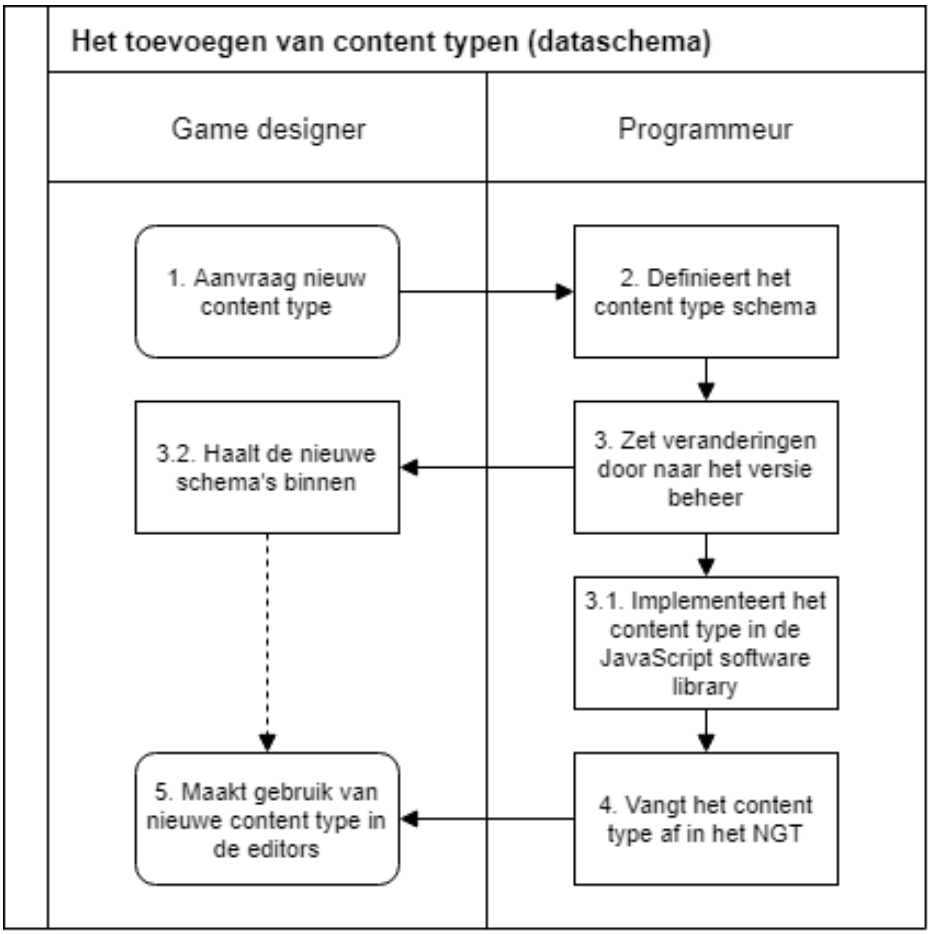
5.3 Dataschema's

Vanwege de diversiteit in game content verschillen de content typen sterk per project. De selectie aan content types wordt geacht om dynamisch te zijn. Deze staan echter 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 5.4. 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 5.5.

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

Figuur 5.4: Pseudo dataschema voor 'sms content'



Figuur 5.5: Een versimpeld proces voor het toevoegen van content typen.

5.3.1 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 5.6. Vervolgens kunnen XML-objecten gevalideerd worden door het schema. Het XML-object in figuur 5.7 is valide, want de structuur komt overeen met die van het dataschema in figuur 5.6. 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 5.6: 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 5.7: Valide XML-object van 'sms content'.

5.3.2 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[57]. Vanwege de vraag naar schema's in JSON is het er een woordenboek opgezet waarmee JSON-schema's opgezet kunnen worden[58]. 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 5.8. Het JSON-document in figuur 5.9 is valide volgens het schema in figuur 5.8.

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

Figuur 5.8: JSON-schema voor 'sms content'.

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

Figuur 5.9: Valide JSON-document van 'sms content'.

5.4 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[59].

5.4.1 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[60]. 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[61].
- Het daadwerkelijk schema waartegen JSON-documenten gevalideerd worden. Een voorbeeld is terug te zien in figuur 5.8.

5.4.2 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 5.10. Deze properties kunnen vervolgens worden (her)gebruikt in de daadwerkelijke content typen, door gebruik van het $ref keyword[60]. Dit keyword refereert naar een ander schema via het schema $id.

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

Figuur 5.10: Voorbeeld content type JSON-Schema.

Met deze properties kan er een basis content schema worden opgezet. Deze bevat properties die in elk concrete content type terugkomen. In figuur 5.11 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 5.11: JSON-schema van base content.

5.4.3 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[60]. Het schema in figuur 5.12 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.

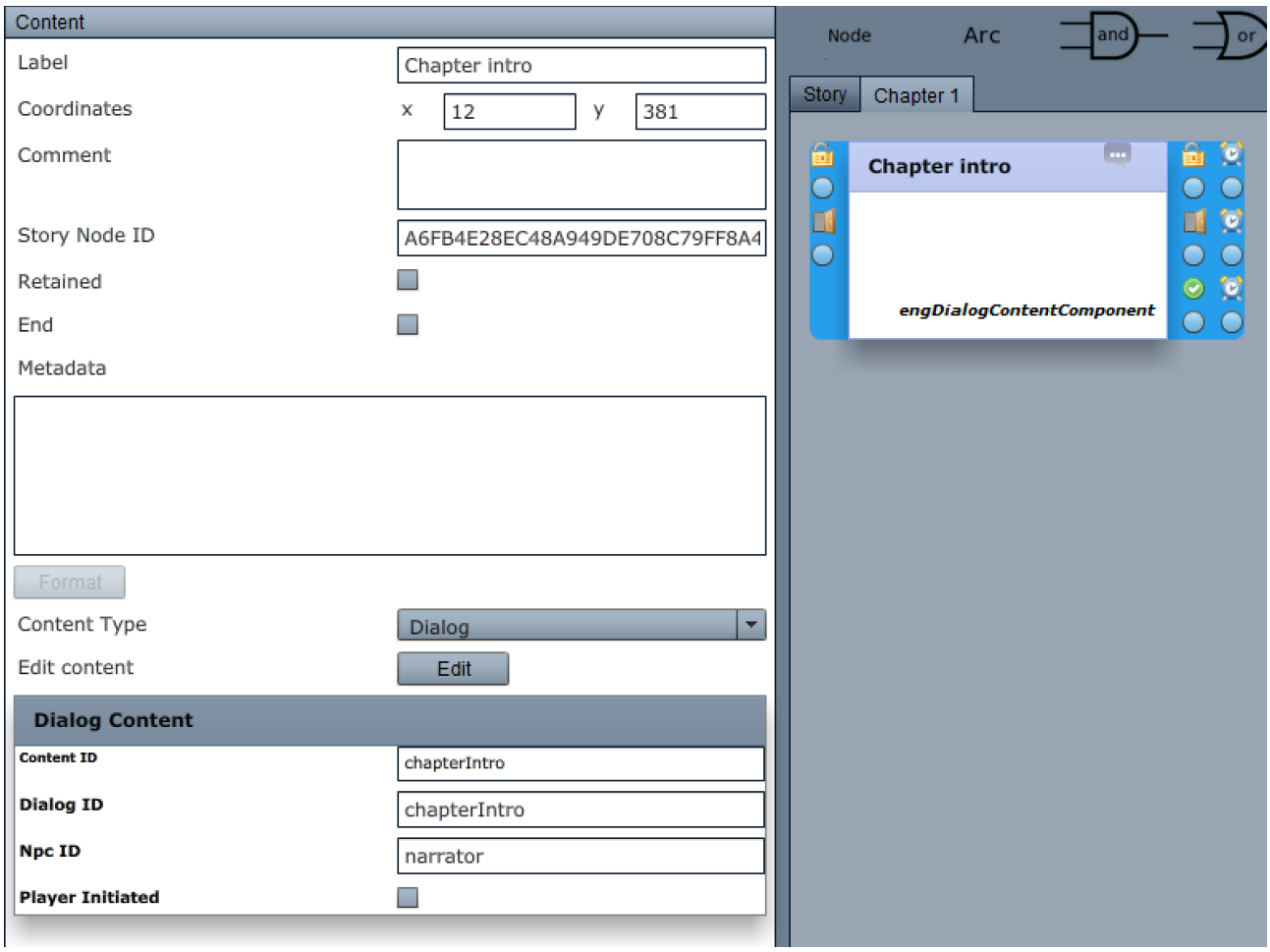
```
{
    "textContent": {
        "allOf": [
            {
                "$ref": "#/definitions/baseContent"
            },
            {
                "$id": "#/contentTypes/textContent",
                "title": "Text content",
                "description": "Textual content",
               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

Figuur 5.12: Een content schema die gebruik maakt van het 'allOf' keyword.

5.5 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 5.13) is

verantwoordelijk voor het tonen van deze properties.



Figuur 5.13: De inspector die een geselecteerde dialog node toont.

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.

5.5.1 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.

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 [62]. 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 5.14. 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.

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "$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 5.14: Een JSON-Schema met referentie.

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[61]. 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.

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

5.5.2 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[60] 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

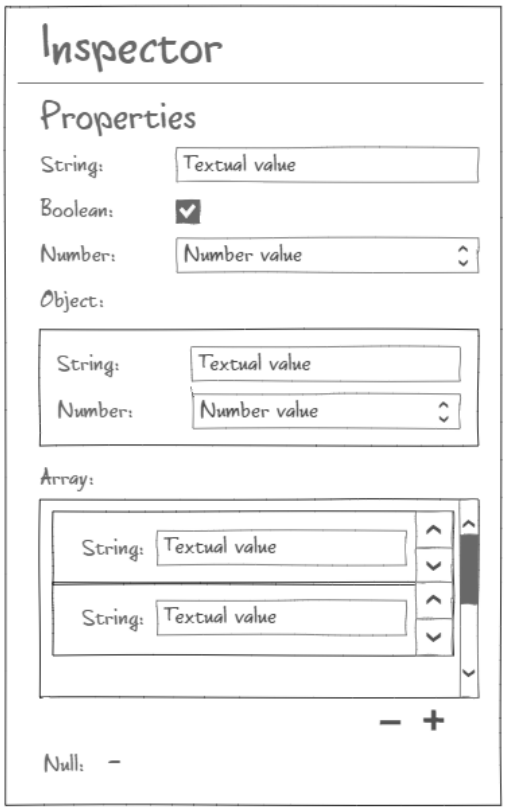
Ieder type heeft een eigen representatie. Met React kan de representatie van ieder type worden ingekapseld in componenten.

In figuur 5.15 wordt een voorstel gedaan voor de representatie van de types[3].

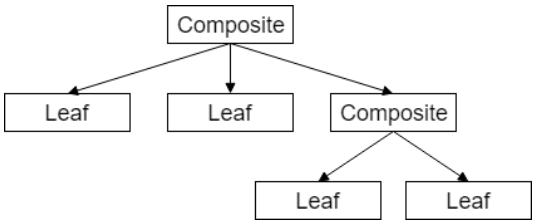
Compositie

Het voorstel in figuur 5.15 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 5.16), zo zijn de types 'object' en 'array' composities. Leaf objecten beheren geen onderliggende objecten. De boom is valide wanneer deze begint met een composite en eindigt met leaves.

[3] "null" heeft geen representatie



Figuur 5.15: Representatie van primitieve JSON-schema types.



Figuur 5.16: Een valide compositie boom.

5.6 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.

5.7 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. In hoofdstuk 7 wordt een voorstel gedaan op een overkoepelende 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 6

Formalismen

In dit hoofdstuk wordt er gekeken naar formalisme binnen interactive story telling. Er wordt een voorstel gedaan op de softwarearchitectuur van de nieuwe editor om meerdere formalisme te ondersteunen in één omgeving. Deze softwarearchitectuur betreft het leggen van een scheiding tussen de editor en het achterliggende formalisme. Het ondersteunen van meerdere formalismen maakt het onderhouden van twee aparte editors overbodig.

6.1 Formalismen binnen interactive story telling

6.1.1 De rol van formalisme

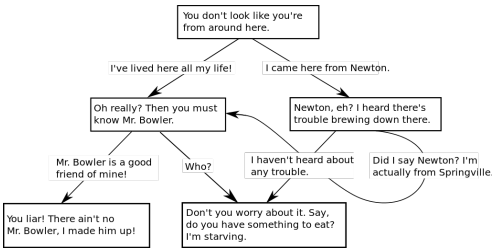
Het formaliseren van het narratief speelt een grote rol tijdens het ontwikkelproces van narrative games. Volgens een vastgesteld formalisme wordt er betekenis gegeven aan de structuur achter het narratief. Een formalisme is een theorie die bepaalde richtlijnen afdwingt en waarde koppelt aan syntax. In figuur 6.1 is een voorbeeld van een formalisme terug te zien. Dit formalisme wordt ook wel een story graph genoemd.

Een formalisme dient als een communicatiemiddel tussen zowel mens als computer. Mensen en computers die kennis hebben van het formalisme interpreteren deze vrijwel hetzelfde. Hierdoor kunnen teamleden efficiënt samenwerken aan een narratief; ze zitten op dezelfde lijn en structureren beide volgens het formalisme.

6.1.2 Formalismen voor verschillende doeleinden

Story graphs

Veel interactieve verhalen met vertakkingen kunnen worden gerepresenteerd als een boom[63]. Deze bestaat uit nodes (ook wel vertices genoemd) die de uiting van de gesprekspartner representeren. Na de evaluatie van deze nodes kan er een keuze gemaakt worden. Deze discrete set van keuzes zijn terug te zien in de story tree als edges, hetgeen dat de nodes met elkaar verbindt. De gemaakte keuze bepaald de volgende node die geëvalueerd zal worden en dus de uitkomst van het verhaal. Dit formalisme laat ons toe om een inzichtelijk en simpele dialoog tussen twee personen te modelleren (figuur 6.1).



Figuur 6.1: Een simpele story tree. [1]

Behaviour tree

// todo

[1] Bron: `https://commons.wikimedia.org/wiki/File:Dialog_tree_example.svg`

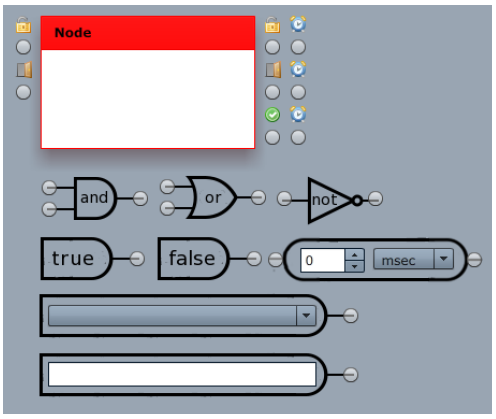
6.1.3 Syntactische vormen

Volgens formalisme kan een narratief uitgedrukt worden in syntactische vormen. Deze hebben verder geen waarde of betekenis, totdat er een interpretatie op losgelaten wordt. Een goed voorbeeld hiervan is terug te zien in de wiskunde.

Zo bestaat de stelling van Pythagoras ook uit syntactische vormen die weergeven worden als letters:

$$a^2 + b^2 = c^2$$

Binnen de wiskunde hebben deze letters volgens het formalisme zelf geen betekenis. Dit wordt onderbouwd door het feit dat letters gebruikt kunnen worden voor andere stellingen, functies en vergelijkingen. De interpretatie op de eigenschappen van letter, zoals positie binnen de stelling, geven betekenis aan deze syntactische vorm.

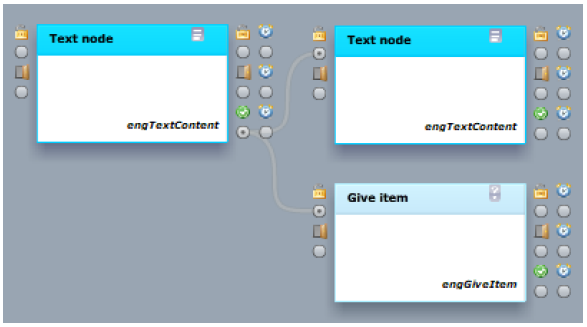


Figuur 6.2: Bouwblokken in de story editor.

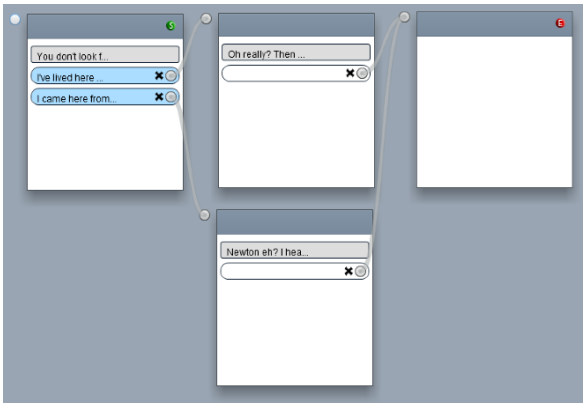
Dit begrip komt ook terug in de editors in de vorm van nodes. Met deze nodes wordt er een narratief geformuleerd die de richtlijnen van het formalisme respecteert. We kunnen ook wel zeggen dat deze syntactische vormen bouwblokken zijn in de editors. In figuur 6.2 zijn enkele bouwblokken die in de story editor voorkomen te zien.

6.2 Formalisme binnen de story- en dialog editor

De story- en dialog editor maken beide gebuikt van een ander formalisme. Dit is duidelijk te zien aan hoe het narratief representeert wordt door de editors (zie figuur 6.3 en figuur 6.4).



Figuur 6.3: Story editor visualisatie van achterliggende data.



Figuur 6.4: Dialog editor visualisatie van achterliggende data.

Aan de huidige code base is te zien dat er een nauwe koppeling bestaat tussen de interface en het achterliggende formalisme. Dit maakt het lastig om meerdere formalismen te ondersteunen.

Dit kan een reden zijn geweest waarom er twee editors zijn opgezet; om de scheiding te maken tussen formalismen. Het is een uitdaging om in één omgeving meerdere formalismen te ondersteunen, omdat deze bestaat uit andere syntactische vormen en gebruik maken van andere principes.

Wel is het wenselijk om meerdere formalismen in één omgeving te ondersteunen om de volgende voorkomende problemen op te kunnen lossen:

- Houdbaarheid; er hoeft nog maar één omgeving onderhouden te worden in plaats van twee wat schilt in kosten.
- Houdbaarheid; generieke editor functionaliteit (copy/ paste, verplaatsen van nodes) kan worden hergebruikt en hoeft maar één keer geïmplementeerd te worden.
- Future proofing; benodigde formalismen kunnen in de toekomst worden toegevoegd. Er hoeft niet nog een nieuwe editor gemaakt te worden.
- Co-creatie; er zou eventueel een versimpeld formalisme toegevoegd kunnen worden waar klanten mee overweg kunnen. Hiernaast kan het versimpelde formalisme gebruikt worden om de volledige flow van het verhaal in kaart te brengen. Bij &ranj is er al onderzoek gedaan naar formalisme voor klant co-creatie [64].

6.3 De scheiding leggen tussen de editor en formalisme

Om een flexibele tool op te zetten voor het vertellen van diverse digitale interactieve verhalen is het nodig om verschillende formalisme voor diverse doeleinden te kunnen gebruiken. Met de toekomst blik van &ranj en het 'bad news' dialoog moet er een formalisme ondersteund worden waarin een meer AI-benadering toegepast kan worden. Om dit mogelijk te maken moet er een manier komen om formalismen toe te kunnen voegen aan de editors.

Om de scheiding te leggen tussen de editor en het achter liggende formalisme zijn de volgende vragen opgesteld:

- Hoe kunnen er bouwblokken worden geformuleerd per formalisme?
- Hoe kunnen er regels worden opgelegd aan het verbinden en het in elkaar voegen van deze bouwblokken?
- Hoe zou een algoritme eruitzien om vanuit een visuele structuur te compileren naar een formalisme?

6.3.1 Het formuleren van bouwblokken

Ieder formalisme wordt ondersteund door syntactische vormen. Een concretere naam die aansluit bij de editors hiervoor is 'bouwblokken'. Gebruikers van de editors zullen deze bouwblokken gebruiken om een narratief te definiëren volgens een vooraf vastgesteld formalisme. De representatie en interpretatie van deze bouwblokken verschillen dan ook per formalisme. Maar voor de editors zijn het allemaal 'nodes' (met eventuele porten) waarmee de node verbonden kan worden aan andere nodes; voor de editor hebben deze nodes geen verdere waarde noch betekenis. Als programmeur moet het mogelijk zijn om op een makkelijke manier bouwblokken te kunnen formuleren. Voor deze use case zijn de volgende requirements opgesteld:

1. Ieder formalisme moet onderbouwd worden door een set aan bouwblokken, zodat de gebruiker deze kan gebruiken om binnen de richtlijnen van het formalisme te werken.
2. Het constructie proces van de bouwblokken moet losstaan van de representatie, zodat verschillende representaties gebruikt kunnen worden voor diverse doeleinden.
3. De representatie van deze bouwblokken moeten losgekoppeld worden van enige waarde of betekenis, zodat deze later volgens het formalisme geïnterpreteerd kunnen worden.

De voornaamste manier om een node te construeren is om de bijbehorende klasse te instantiëren. Dit geeft ons een node object waarvan de representatie statisch is; het instantiëren van de node klassen resulteert altijd in hetzelfde object. Dit schendt requirements 2 en 3. Om constructie van representatie los te koppelen, en zo te voldoen aan de tweede requirement, kunnen er parameters de constructor van de node klasse gedefinieerd worden die de eigenschappen en representatie van de node beschrijven. In figuur 6.3 is een van de bouwblokken in het formalisme van de story editor terug te zien: de ContentTypeNode. Dit bouwblok heeft een label en 8 porten. Het zojuist beschreven constructie proces voor deze node zou er als volgt uit kunnen zien:

Functie handtekening

```
1 constructor Node(label: string, markup: string, allowBodyConnections: boolean,
      ports: Port[]): Node
```

Voorbeeld

```
1 new Node('ContentType', '<rect class="node-body" width="100" height="100"><text
      /></rect>', false, [new Port(...), new Port(...), ...);
```

Deze manier van construeren heeft de volgende consequenties:

- Een node moet geconstrueerd worden met argumenten die voldoen aan de parameters van de constructor.
- Om achter de betekenis van de argumenten te komen moet er gekeken worden naar de constructor handtekening; argumenten zijn niet expliciet.
- Het stimuleert een nauwe koppeling tussen de node klasse en de achterliggende 'diagramming library' die de nodes visualiseert.
- Het stimuleert het aanroepen van het constructie proces op diverse plekken in de code.
- Er kunnen nodes worden gemaakt met verschillende representaties, door verschillende argumenten mee te geven.

Deze manier van construeren wordt al gauw ingewikkeld en onduidelijk ondervonden. Vooral omdat het constructie proces niet leesbaar en expliciet genoeg is. Daarnaast zal het constructie proces op verschillende plekken plaatsvinden, omdat er geen klasse bestaat die hier verantwoordelijk voor is. Dit heeft een grote invloed op de houdbaarheid van het product. Als er later een aanpassing aan een bouwblok gedaan moet worden zal dit relatief veel werk kosten en mogelijk fouten introduceren. Het constructie proces moet geabstraheerd worden, zodat het system onafhankelijk is van hoe de node objecten geïnstantieerd en gerepresenteerd worden. Door gebruikt te maken van 'Creational design patterns' kan er een abstractie laag worden geplaatst waardoor de gezochte scheiding kan worden gelegd[65].

Er is een kleine selectie aan patronen opgesteld die in aanmerking komen om het eerder beschreven probleem op te lossen. Deze design patterns zullen naast het probleem gelegd worden om de toepasselijkheid en consequenties in kaart te brengen. Uiteindelijk zal er een conclusie getrokken worden waarin het best passende design pattern ter sprake komt. De selectie aan design patterns die besproken zal worden bestaat uit:

- Abstract factory
- Builder

Abstract factory

Intentie

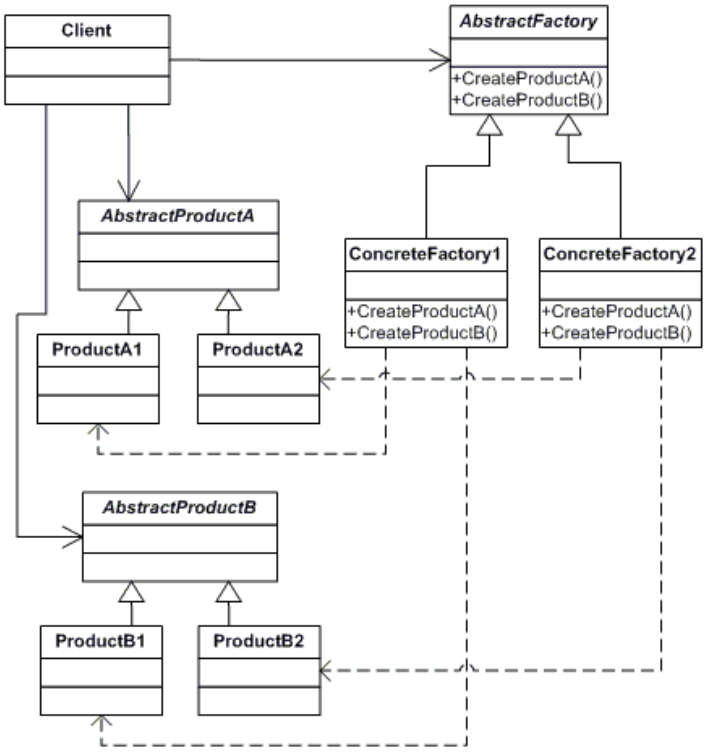
De intentie van dit patroon wordt beschreven als volgt[65][66]:

> *"Provide an interface for creating families of related or depent objects without specifying their concrete classes."*

Toepasselijkheid

Dit patroon helpt de editor onafhankelijk zijn van de manier waarop de nodes geconstrueerd en gerepresenteerd worden [65], wat inhaakt op een deel van de tweede requirement. Verder is het wenselijk om meerdere formalismen te ondersteunen. Hierbij spelen bouwblokken een grote rol: deze ondersteunen het gebruik van een formalisme binnen de editor. Volgens de eerste requirement moet de editor een set aan bouwblokken kunnen aanbieden die de gebruiker kan gebruiken. Het abstract factory patroon maakt het mogelijk om families van nodes te creëren [65]. Hierbij is de familie het formalisme.

Structuur



Figuur 6.5: Klasse diagram van het abstract factory pattern. [2]

Deelnemers

- **AbstractFactory** (NodeFactory)
  - Biedt een interface aan met methodes voor het maken van node objecten.
- **ConcreteFactory** (StateMachineNodeFactory, BehaviourTreeNodeFactory)
  - Implementeert de 'NodeFactory' om concrete node objecten te kunnen maken.
- **AbstractProduct** (Node, mogelijke extra abstractie laag: StateMachineNode, BTNode ...)
  - Biedt de interface voor een type van concrete node objecten.
- **ConcreteProduct** (ContentNode, SelectorNode, ...)
  - Implementeert 'Node' en maakt onderscheid tussen verschillende type nodes mogelijk.
  - Definieert een product dat gemaakt zal worden door het desbetreffende concrete factory object.
- **Client**

[2] Bron: `http://www.dofactory.com/net/abstract-factory-design-pattern`

– Gebruikt de interfaces die gedefinieerd zijn door de 'AbstractFactory' en de 'AbstractProduct' klassen.

Consequenties

- Het isoleert de concrete node klassen; de factory heeft de verantwoordelijkheid en het proces om nieuwe nodes te maken.
- Het maakt het verwisselen van verschillende formalisme makkelijk; Elk formalisme zou een eigen factory kunnen hebben. De factory wordt op één plek in de code geïnstantieerd en omdat de concrete factories allemaal dezelfde interface implementeren kunnen we makkelijk van concrete factory verwisselen.
- Het dwingt het implementeren van interface af; elke concrete node zal de node interface moeten implementeren. Om subklasse specifieke operaties aan te kunnen roepen zal er gebruik gemaakt moeten worden van casting.
- Nieuwe nodes toevoegen is lastig; per type node zal er een methode bijkomen in zowel de abstract factory als de concrete factories. Hiernaast kan dit zorgen voor concrete factories die operaties bevatten met een lege body, omdat bepaalde type nodes niet van toepassing zijn in het formalisme dat de concrete factory ondersteund.

Conclusie
Het is wenselijk om het constructie proces te isoleren en te scheiden van de rest van de applicatie. Dit maakt het makkelijker om later eventueel van diagramming library te wisselen en nodes aan te passen. Verschillende implementaties van 'AbstractFactory' kunnen ieder nodes voor een formalisme aanbieden. Echter omdat elke 'ConcreteFactory' de 'AbstractFactory' implementeert zullen deze moeten voldoen aan de interface van de geabstraheerde factory. Dit kan zorgen voor lege methodes die verder geen waarde hebben. Hiernaast kan dit de illusie wekken dat elke node terugkomt in ieder formalisme. Tenslotte is het toevoegen van nieuwe nodes lastig. Volgens requirement 1 moet er een set aan bouwblokken aangeboden worden. Als er een nieuwe bouwblokken in de vorm van nodes aan de set toegevoegd moeten worden zal de interface van de 'AbstractFactory' ook aangepast moeten worden. Dit heeft als gevolg dat iedere 'ConcreteFactory' deze nieuwe methode ook moet implementeren. Hoe meer formalismen er zijn, des te moeilijker het zal zijn om nieuwe bouwblokken toe te voegen. Dit maakt het 'Abstract Factory' design pattern ongeschikt voor deze use case.

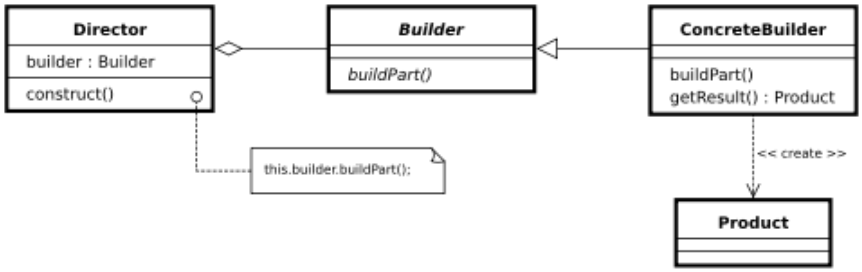
Builder

Intentie
De intentie van dit patroon wordt beschreven als volgt[65][67]:

"Separate the construction of a complex object from its representation so that the same construction process can create different representations."

Toepasselijkheid
Het builder patroon is interessant omdat nodes een vrij complex object zijn die meerdere representaties zal hebben in de editor. Er moeten verschillende bouwblokken ondersteund worden (requirement 1) die ieder een eigen presentatie hebben (requirement 2). Hiernaast maakt het patroon het constructie proces expliciet door een interface aan te bieden waarmee nodes opgebouwd kunnen worden.

Structuur



Figuur 6.6: Klasse diagram van het builder pattern. [3]

Deelnemers

- **Builder** (NodeBuilder)
  - Biedt een interface voor het maken van node onderdelen (e.g. porten, labels, visuals).
- **ConcreteBuilder** (StateMachineNodeBuilder, BehaviourTreeNodeBuilder)
  - Implementeert de 'Builder' interface en maakt het mogelijk om onderdelen te maken.
  - Definieert en houdt bij hoe de representatie gemaakt wordt.
  - Biedt een interface om de uiteindelijke node te verkrijgen.
- **Director** (StateMachineDirector, BehaviourTreeDirector)
  - Gebruikt de 'Builder' interface om de verschillende bouwblokken per formalisme te construeren.
- **Product** (ContentNode, SelectorNode, ...)
  - Representeert de node waaraan de 'ConcreteBuilder' bouwt.
  - Bevat de aangebouwde onderdelen en achterliggende model (e.g. content type of een expressie).

Consequenties

- Het isoleert het constructie proces en de representatie van nodes; de node editor hoeft niet te weten hoe een node gemaakt en gerepresenteerd wordt. De builder biedt een interface voor het construeren van nodes die de concrete builder implementeert. Deze code om nodes te creëren hoeft maar één keer geschreven te worden. Directors kunnen deze interface gebruiken om verschillende varianten van nodes te maken.
- Het constructie proces van nodes is explicieter; de builder biedt een interface met methodes die direct beschrijven welk onderdeel er gemaakt wordt. Zo is de constructie die plaats vindt in figuur 6.7 explicieter dan figuur 6.8.
- Fijnere controle over constructie proces van nodes; nodes worden stap voor stap in elkaar gezet. In tegenstelling tot andere creational design patterns worden nodes niet in één keer gemaakt.

Conclusie

Het builder patroon biedt de mogelijkheid om nodes stapsgewijs te construeren. Door middel van de interface die de 'NodeBuilder' klasse biedt kunnen nodes expliciet en modulair opgebouwd worden (zie figuur 6.7). Dit maakt het makkelijk om nieuwe bouwblokken te definiëren, zonder iets af te weten van hoe de nodes gemaakt worden. Deze code is één keer geschreven en bevind zich in de concrete builders. De 'Director' klasse kan geabstraheerd worden die per formalisme geïmplementeerd kan worden. Deze concrete directors kunnen een set aan bouwblokken opbouwen die makkelijk aan te passen is, waarmee voldaan wordt aan de eerste requirement. Het patroon biedt ook een fijnere controle over het constructie proces, omdat nodes stapsgewijs op gebouwd kunnen worden. Dit maakt het makkelijk om bouwblokken met verschillende representaties te creëren en hiermee wordt voldaan aan de tweede requirement.

[3] Bron: `https://en.wikipedia.org/wiki/Builder_pattern`

```
return this._builder
    .build<ContentNode>()
    .label('ContentNode')
    .addPort(CompletedPort)
    .addPort(UnlockedPort)
    .getNode();
```

Figuur 6.7: Constructie stap voor stap door de 'Director', geschreven in TypeScript.

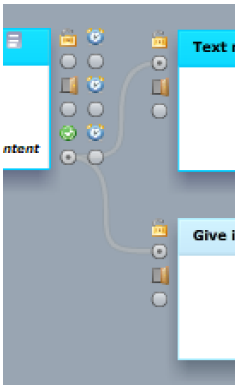
```
return new Node('ContentNode', [new Port(...)]);
```

Figuur 6.8: Directe instantiatie door constructor.

6.3.2 Restricties opleggen aan het verbinden en in elkaar voegen van bouwblokken

Door de bouwblokken aan elkaar te kunnen verbinden kunnen er een of meerdere verhaallijnen worden gemoduleerd. Verschillende bouwblokken worden met elkaar verbonden om volgorde en relatie tussen elkaar aan te geven.

Formalismen leggen restricties op zowel het verbinden als het in elkaar voegen van bouwblokken. Dit kan duidelijk teruggezien worden in de huidige editors. Hierin wordt er gebruik gemaakt van porten om de nodes met elkaar te verbinden (figuur 6.9). Zowel de story- als dialog editor lijkt gebruik te maken van semantische port groepen. Dit maakt het mogelijk om een betekenis te geven aan de porten. Beide hebben een 'in' en 'uit' groepen, waarbij de er geen verbinden kunnen worden gevormd tussen dezelfde groep; 'in' kan niet verbonden worden met een andere 'in'. Hiernaast werken sommige porten met tijd (zie de porten met klokjes in figuur 6.9) en andere met logische expressies.



Figuur 6.9: Verbonden porten in de huidige story editor.

In de dialog editor kunnen bouwblokken in elkaar gevoegd worden (zie figuur 6.10). Hier zitten wel bepaalde regels aan vast. Zo kunnen er alleen 'commands' (1) en 'options' (2) gevoegd worden in een 'state' (3). De 'command' en 'option' bouwblokken kunnen alleen bestaan als ze gevoegd zijn.

Het valideren van verbindingen

Niveau

Eerst moet er gekeken worden naar op welke niveau gevalideerd moet worden. Het niveau bepaald de granulariteit waarop de controle uitgevoerd wordt. Als er gekeken wordt op groepsniveau is het makkelijker om de rule set, of terwijl de verschillende restrictie regels, overzichtelijk en houdbaar in te delen. Hiertegenover staat dat er sprake is van een validatie op hoog niveau. Er wordt gekeken naar semantische groepen en niet naar de individuele porten. Dit maakt het moeilijk om eventuele uitzonderingen in de restricties toe te laten. Veel uitzonderingen kunnen echter weer zorgen voor onoverzichtelijkheid.

Figuur 6.10: Dialog editor: bouwblokken gevoegd in andere bouwblokken.

Het vastleggen van een rule set op groep niveau bevorderd de houdbaarheid, maar maakt het systeem ook minder flexibel. Het uitgangspunt van dit onderzoek is om alles zo flexibel mogelijk op te zetten, zolang dit geen hevige negatieven gevolgen heeft. Daarom zal het werken met semantische port groepen worden gestimuleerd, maar wanneer er een fijnere maten van controle vereist is moet het mogelijk zijn om op individueel niveau te valideren. Hierbij zal de validatie op individueel niveau de

groepsvalidatie overschrijven.

Validatie

Om volledige controle te geven over het validatie proces zal het systeem gebruik maken van een validatie functie. Deze functie zal worden uitgevoerd wanneer de gebruiker een verbindingen gaat maken. De validatie functie kan uitgedrukt worden als:

validate: (x) => boolean

Hierbij is 'x' het gene waar de verbinding tegen gevalideerd wordt. Omdat formalismen directe verbindingen met andere porten zouden kunnen toelaten, en het systeem hier geen restricties op moet leggen, kan 'x' zowel een port als een node zijn. Dit introduceert een nieuw probleem, omdat we niet weten wat voor type object meegegeven wordt als argument. Het maken van een validatie functie is lastig als het niet duidelijk is waartegen gevalideerd gaat worden. Echter is dit in een statically typed language, een programmeertaal waarin variabelen bestaan uit een type en een waarde, op te lossen door middel van method overloading. Hierbij wordt er voor elke mogelijke input een methode gemaakt (in dit geval twee methodes). Tijdens het compileren wordt er beslist welke methode waar gebruikt wordt. In een dynamically typed language bestaat het concept method overloading niet, omdat variabelen niet gebonden zijn aan een type. Dit betekent dat Typescript, wat uiteindelijk omgezet wordt in Javascript (een dynamically typed language), geen gebruik kan maken van method overloading. Echter biedt Javascript een uitweg met de 'instanceof' operator. Deze operator geeft een boolean, een logische waarde, terug die waar zal zijn als de constructor van het type terugkomt in de prototype chain. Een voorbeeld van een validatie functie geschreven in Typescript is terug te zien in figuur 6.11. Deze functie beschrijft dat er alleen een verbinding mag worden gelegd met andere Nodes. Als er een validatie functie gedefinieerd is op individueel niveau zal deze de groepsvalidatie functie overschrijven.

```
1  validateConnection(against: Port | Node): boolean => {
2      if (against instanceof Port) {
3          return false;
4      }
5
6      if (against instanceof Node) {
7          return true;
8      }
9  }
```

Figuur 6.11: Een voorbeeld van een validatie functie in Typescript.

Het valideren van het in elkaar voegen

Dezelfde methode en techniek kan gebruikt worden ter validatie van het in elkaar voegen van nodes.

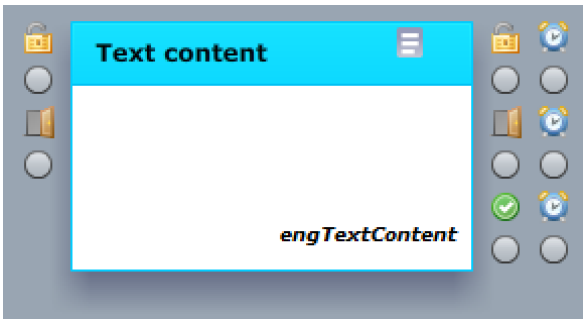
6.3.3 Een algoritme om vanuit een visuele structuur te compileren naar een formalisme

Nadat het verhaal geschreven is in de editor wordt deze geëxporteerd zodat het NGT hiervan gebruik kan maken. Dit exporteer proces zet de visuele structuur om naar JavaScript Object Notation (JSON), een gestandaardiseerd dataformaat. Het NGT kan dit formaat uitlezen en omzetten en interpreteren. Tijdens dit compilatie proces wordt editor specifieke informatie, zoals de positie en uiterlijk van nodes, niet mee geëxporteerd. Voor het NGT heeft deze informatie geen waarde en is dus overbodig.

Het scheiden van bouwblokken en waarde

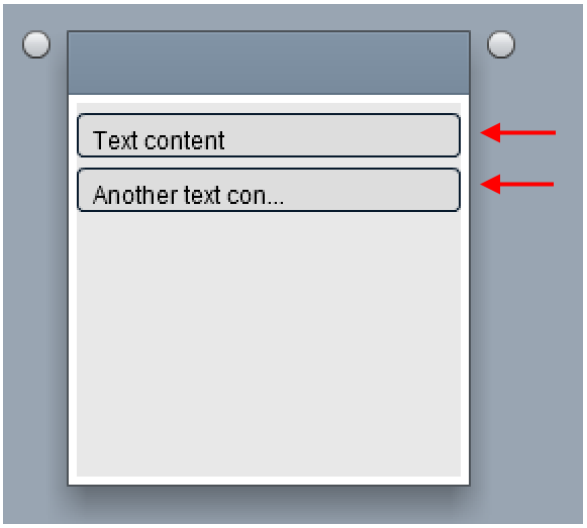
In de huidige editors zit het formalisme vast gebakken in de visuele structuur. Bouwblokken hebben naast een uiterlijk ook een methode die een JavaScript object met data vrijgeeft. Dit object wordt vervolgens direct geserialiseerd; omgezet naar een formaat dat opgeslagen kan worden. De editor koppelt dus waarde en formalisme aan de bouwblokken zelf. Wat resulteert in niet herbruikbare bouwblokken; de bouwblokken zijn direct gekoppeld aan het formalisme.

Om het hergebruik van bouwblokken tussen meerdere formalismen mogelijk te maken moet de waarde van de bouwblokken gescheiden worden. Bouwblokken kunnen theoretisch terugkomen in meerdere formalismen waarin ze anders geïnterpreteerd moeten worden. Zo kunnen content typen terugkomen in zowel het formalisme van de story- als dialog editor. In de story editor zijn content typen terug te zien in de vorm van nodes, met verschillende in en uit porten (figuur 6.12). De dialog editor gebruikt content typen zonder porten (figuur 6.13). Hier zijn de content typen ingevoegd in andere nodes.



Figuur 6.12: Een content type in de story editor.

In alinea 6.3.1 wordt representatie van bouwblokken losgekoppeld van de werking door gebruik te maken van het 'builder' design pattern. De volgende stap is om de visuele structuur om te zetten naar een export-bestand volgens de richtlijnen van het desbetreffende formalisme. Hiervoor moet de serialisatie logica losgetrokken worden van de nodes zelf, zodat nodes geïnterpreteerd kunnen worden volgens het formalisme. Wanneer het verhaal wordt gecompileerd moet er volgens de richtlijnen van het formalisme waarde gekoppeld worden aan de bouwblokken. Hiernaast moet er worden gekeken hoe de graaf doorlopen gaat worden.



Figuur 6.13: Een content type in de dialog editor.

Compilatie stap

De interpretatie van nodes moet losgekoppeld worden van de visuele structuur. Dit betekent dat de nodes zelf geen methode moeten bevatten die de data van een node vrijgeeft. Nodes moeten geïnterpreteerd worden volgens zowel interface als formalisme.

Door gebruikt te maken van het visitor design pattern wordt er een context aangeboden waarin het exporteer-bestand opgebouwd kan worden. De 'gang of four' koppelt de volgende intent aan dit design pattern[65]:

"Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without chaning the classes of the elements on which it operates."

Deze intentie kan gekoppeld worden aan het eerder beschreven probleem. De operatie die uitgevoerd wordt op de elementen van een object structuur, betreft het omzetten van de visuele structuur naar een export-bestand. Hierbij worden operaties gedefinieerd in de visitor in plaats van in de nodes zelf waarmee interpretatie van de nodes losgekoppeld wordt.

Visitor pattern

Het visitor pattern definieert de volgende deelnemers:

- **Visitor** (ExportGeneratingVisitor)

– Definieert voor iedere concrete element klasse in de datastructuur een 'Visit' methode. Echter werken we met JavaScript, een programmeertaal die loosly typed is; data typen worden niet expliciet gedefinieerd. Hierdoor kan er geen gebruik worden gemaakt van method overloading. Dit heeft als gevolg dat de Visitor interface maar één 'Visit' methode zal definieren. In de concrete visitor zullen de type specifieke operaties worden gedefinieerd.

- **Concrete Visitor** (StoryExportGeneratingVisitor)

– Implementeerd de 'Visitor' interface en biedt content voor het algoritme. Hiernaast kan de concrete visitor staat verzamelen, in dit geval bouwt deze het export-bestand op.

- **Element** (Node)

– Definieert een 'Accept' methode met de Visitor als parameter.

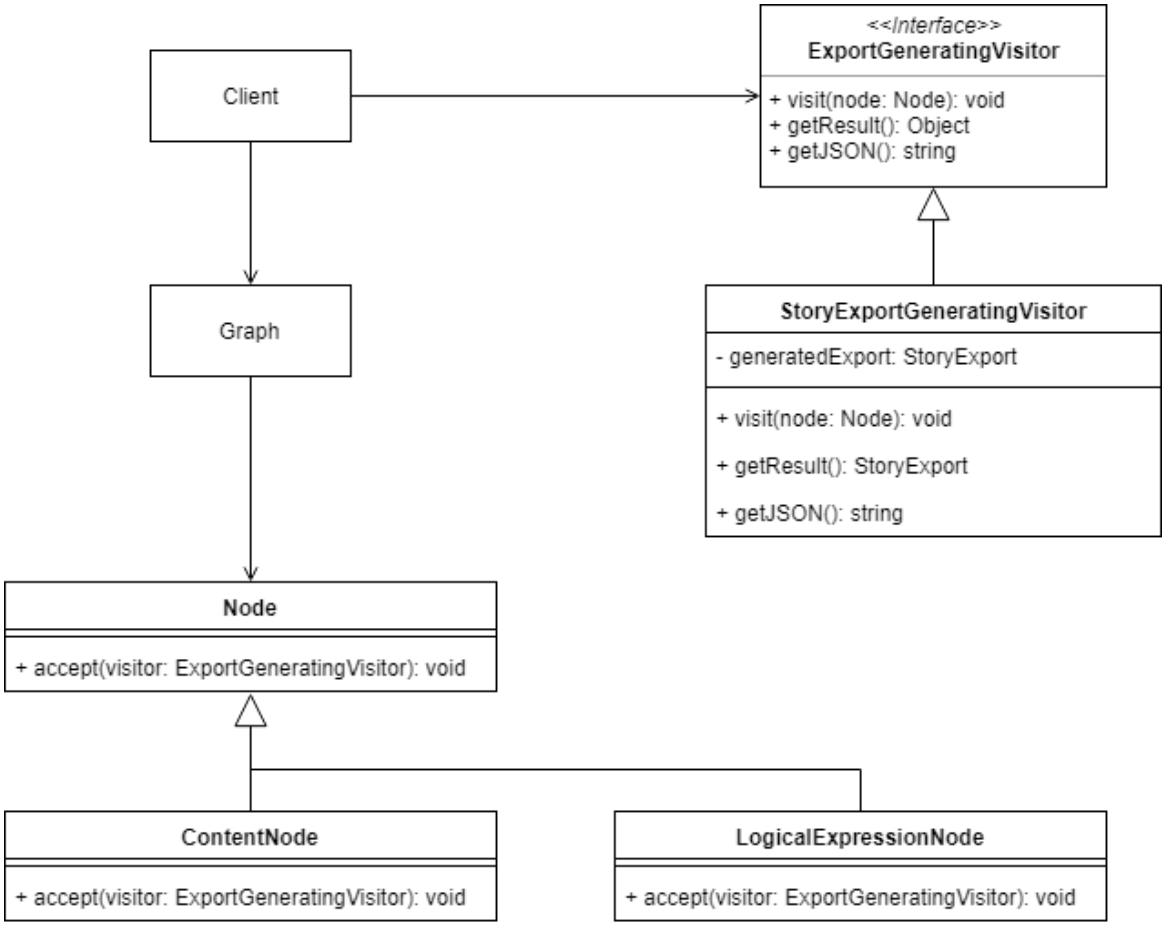
- **ConcreteElement** (Bouwblokken in formalisme, zoals 'ContentNode')

– Implementeert de 'Element' interface, met de 'Accept' methode die een visitor as parameter neemt.

- **ObjectStructure** (Graph)

– Biedt een interface om te itereren over haar elementen.

Het totaal plaatje is terug te zien in figuur 6.14. Voor ieder formalisme wordt een concrete visitor aangemaakt, zodat ieder formalisme de elementen kan interpreteren volgens haar eigen richtlijnen.



Figuur 6.14: Het visitor pattern toegepast voor het formalisme van de story editor.

Consequenties

Het gebruik van het visitor pattern om van visuele structuur naar exportbestand te compileren heeft de volgende consequenties:

1. Bouwblokken kunnen worden toegevoegd aan een formalisme, zonder zelf van het formalisme af te weten. Hiermee wordt vervuiling van de nodes voorkomen.
2. Bouwblokken kunnen worden hergebruikt in andere formalisme. De concrete visitor voor het desbetreffende formalisme moet dan wel een methode definiëren die de bouwblokken kan bezoeken.

3. Elke bouwblok moet een 'Accept' methode implementeren.

4. Concrete visitors kunnen staat bijhouden. Dit wordt gebruikt om het export-bestand op te bouwen.

In een strongly typed programmeertaal zou het toevoegen van concrete elementen lastiger zijn. Hiervoor moet de interface van de visitor aangepast worden om het nieuwe element te ondersteunen. Dit zou ook betekenen dat iedere concrete visitor (en dus elk formalisme) een visit methode zou moeten implementeren voor ieder bouwblok. Als bouwblokken niet worden gebruikt in een formalisme zou dit lege methode opleveren. Echter wordt deze consequentie ontweken doordat er gewerkt wordt in JavaScript, een loosly typed programmeertaal.

Het prototype implementeert het visitor pattern en valideert zo het gebruik van het visitor pattern voor de compilatie stap.

6.4 Conclusie

Met dit hoofdstuk wordt de volgende deelvraag beantwoord: "Hoe kan de architectuur achter de editor zo worden opgezet dat er in latere stadia nieuwe narratieve formalismen makkelijk doorgevoerd kunnen worden?".

Een formalisme bestaat (1) uit syntactische vormen legt (2) restricties op het verbinden en het in elkaar voegen van deze vormen. Met behulp van het builder design pattern kunnen bouwblokken worden gescheiden van representatie. Dit laat het hergebruik van bouwblokken met verschillende representaties toe. Hiernaast biedt de builder een expliciete interface voor het construeren van nodes en kapselt deze het constructieproces af. Ieder formalisme beschikt over een eigen director die een set aan beschikbare bouwblokken formuleert voor het desbetreffende formalisme.

Restricties tussen het verbinden van deze syntactische vormen kunnen worden opgelegd door gebruik te maken van een validatie functie. Wanneer de bouwblokken gebruik maken van porten worden er semantische port groepen gedefinieerd om restricties op groepsniveau toe te laten. Restricties op individueel niveau overschrijven de restricties gedefinieerd op groepsniveau.

De visuele weergave van bouwblokken kan worden losgekoppeld van één ingebakken formalisme door gebruik te maken van het visitor design pattern. Hierdoor worden bouwblokken en waarde losgekoppeld. Tijdens het compileren van de visuele structuur naar een export-bestand zal er waarde worden gehangen aan de bouwblokken volgens de richtlijnen van het desbetreffende formalisme. Dit laat het hergebruiken van bouwblokken tussen verschillende formalismen toe.

Door formalisme specifieke logica los te koppelen van de visuele weergave van bouwblokken kunnen er in latere stadia nieuwe bouwblokken geformuleerd worden en nieuwe formalisme doorgevoerd worden.

Hoofdstuk 7

Overkoepelende Projectstructuur

In dit hoofdstuk wordt er een voorstel gedaan om de editors deel uit te laten maken van een overkoepelende projectstructuur; de editors zullen het gehele project openen in plaats van een enkel bestand. Door de editors beschikbare bestanden te laten beheren kunnen er sterkere referenties worden gemaakt tussen de game content en de editor. Deze referenties laten gewenste functionaliteiten toe die eerst niet mogelijk waren. Om het beheren van bestanden mogelijk te maken wordt er een voorstel gedaan om de huidige folderstructuur van het NGT te veranderen. Verder worden er twee semantische lagen geïntroduceerd die de foutgevoeligheid van de nieuwe editor verminderd.

7.1 Projectstructuur narrative game

Narrative game projecten wordt ontwikkeld in het NGT. Dit template dwingt een folderstructuur af die vrijwel ieder project aanhoudt. In deze folderstructuur bevindt zich de code en content voor de game. De bestanden die samen de game content omschrijven worden ook wel 'assets' genoemd. Voorbeelden van assets zijn: afbeeldingen, video's, geluidsbestanden en configuratie bestanden. Kortom, alle media en data die het spel (in)direct aan de speler toont. Vaak worden deze asset al geordend in mapjes wat al wijst op een behoefte aan een projectstructuur.

7.2 Koppeling tussen assets en de editors

De editors maken gebruik van deze assets. Zo kan een gebruiker een verhaal schrijven waarin dialogen bestaan uit teksten en een achtergrond afbeelding. Referenties naar assets worden beschreven in een JSON-bestand. Hierin wordt het pad naar de asset gelinkt aan een sleutel (bijlage A). In de editor wordt vervolgens de sleutel gebruikt om te refereren naar asset.

Dit betekent dat de editors niks af weten van de assets in het project. Er is een folder vol assets die allemaal worden ingeladen. Deze selectie ingeladen assets kunnen worden verwerkt in het verhaal. Er zijn use cases voor een overkoepelende projectstructuur opgesteld en gevalideerd in de wekelijkse meetings met de bedrijfsbegeleider en opdrachtgever (bijlage B). Hieruit blijkt dat het gebruikt van onbeheerde assets invloed heeft op de foutgevoeligheid, efficiëntie en functionaliteit van de editors.

7.2.1 Foutgevoeligheid

Content typen met referenties naar assets, zoals 'image content' welke refereert naar een image asset, zijn erg gevoelig voor menselijke fouten. Deze content typen tonen voor iedere referentie een invoerveld die één lijn aan tekst bevat. Hierin kunnen gebruikers de sleutel invoeren die bij de juiste asset hoort. De invoer kan niet worden gecontroleerd door de editor wat twee problemen introduceert. Typefouten of referenties naar niet bestaande assets worden toegelaten. Hiernaast wordt het invoeren van verkeerde typen assets toegestaan. Zo zou 'image

content' een referentie kunnen bevatten naar een mp3-bestand. De editor kunnen geen semantiek toewijzen aan assets.

Beide scenario's worden niet afgevangen in de editors en kunnen resulteren in ongewenst gedrag en mogelijke crashen in de game. Dit hakt in op de efficiëntie van de editors.

7.2.2 Inefficiëntie

De foutgevoeligheid haakt in op de efficiëntie van de editors. Door een foutgevoelige editor moeten er meer testen en eventuele correcties uitgevoerd worden om een stabiele game aan te kunnen leveren. Hiernaast is er geen mogelijkheid om ongebruikte assets te filteren, ongebruikte assets blijven altijd in de game zitten. Sommige hiervan zullen zelfs worden ingeladen als ze in het JSON-bestand staan. Dit zorgt voor een langere en onnodige laadtijd van de game.

7.2.3 Disfunctionaliteit

Zoals beschreven in hoofdstuk 4, speelt klant co-creatie een rol in de toekomst van de editor. Hiervoor is het belangrijk dat de editors beschikken over een sterke visualisatie van het verhaal[64]. Dit is lonend voor zowel klant als game design. Deze visualisatie moet naast overzicht ook inzicht bieden in de game content, zo zal 'image content' een preview moeten laten zien van de gerefereerde afbeelding. Om dit mogelijk te maken moet er enigszins sprake zijn van een overkoepelende projectstructuur waarbij assets beheerd worden door de editor.

7.3 Overkoepelende projectstructuur

In een overkoepelende projectstructuur wordt de gehele project folder geopend in plaats van een specifiek bestand. Door de project folder te openen kunnen assets binnen het project geïnventariseerd en beheerd worden. Hiernaast kan de inventarisatie aangepast worden als er assets verwijderd of toegevoegd worden. In paragraaf 7.5 wordt er dieper in gegaan op het beheren van assets.

Bij het openen van de gehele project folder is het alleen wenselijk om assets te beheren die verwerkt kunnen worden in het verhaal. Bestanden die broncode of configuraties voor command line tools representeren kunnen niet worden verwerkt in het verhaal. De editor hoeft niks van deze bestanden af te weten. In de huidige folderstructuur van het 'narrative game template' (NGT), het raamwerk waarin narrative games gerealiseerd worden, bevinden zich bruikbare en niet bruikbare bestanden in deze zelfde folder. Om een gestructureerde overkoepelende projectstructuur toe te laten zullen er aanpassingen moeten worden gedaan op de folderstructuur. Deze aanpassing betreffen alleen de overkoepelende projectstructuur.

7.4 Een ondersteunende folderstructuur

De folderstructuur van het NGT vereist enkele aanpassingen om te voorkomen dat niet bruikbare assets wordt beheerd door de editors. Deze aanpassingen resulteren in één directory met assets die verwerkt kunnen worden in de editors.

In **??** wordt de folderstructuur van het NGT weergeven. De 'src' (source) directory bevat de broncode en game content van de game. Hierin bevinden zich een 'assets' en een 'ranj' directory. De 'ranj' directory bevat de broncode van het spel en zal dus geen deel uit moeten maken van de overkoepelende projectstructuur. In de 'assets' directory zitten, zoals de naam luidt, alle asset die worden verwerkt in de game. Hiernaast bevinden zich project specifieke configuraties en flash files[1] ook in deze directory. Deze project specifieke configuraties en flash files zijn niet verwerkbaar in het verhaal en moeten dus uit de assets directory. Project specifieke configuraties zouden eventueel wel gebruikt kunnen worden door de editors. Een voorbeeld hiervan is het content typen dataschema beschreven in hoofdstuk 5.

Dit creëert de behoefte voor een nieuwe directory genaamd 'ProjectSettings'. Hierin zullen zich project specifieke configuratie bestanden bevinden. Waarbij sommige bestanden, zoals het content typen dataschema,

[1] Visual designers gebruikt Adobe Animate CC om schermen te ontwikkelen voor narrative games.

uitgelezen zal worden door de editors. Dit maakt het definiëren van een selectie aan project specifieke content typen op projectniveau mogelijk.

7.5 Het beheren van assets

Om de assets te beheren zal de editor een lijst moeten bijhouden met daarin informatie over beschikbare assets in de 'assets' directory. Elementen in deze lijst worden gerepresenteerd als:

$$\{\ id : string,\ path : string,\ type : string\ \}$$

Waar;

id bestaat uit een unieke string die eenmaal gegeneerd zal worden wanneer de asset geïnventariseerd wordt. Deze wordt gebruikt om naar assets te refereren. Dit speelt een rol in het exporteer proces beschreven in paragraaf 7.7.

path een relatief pad naar de asset vanaf de root van het project is. Zo kan zowel de editor als het NGT de locatie van de asset achterhalen. Wanneer er naar deze asset gerefereerd wordt kan de bestandsnaam uit dit pad getoond worden om inzicht te geven in de gerefereerde asset.

type semantiek toekent aan de asset (e.g. 'image', 'sound').

Door gebruik te maken van een 'id' blijven referenties intact wanneer het pad naar het bestand veranderd. Als de asset verwijderd wordt zullen referenties naar deze asset breken. De editor kan vervolgens aangeven welke referenties gebroken zijn en eventueel vragen met welke asset deze vervangen moeten worden. Een nadeel aan het gebruik van een onleesbare sleutel is dat de assets niet statisch opgehaald kunnen worden in het NGT. Het NGT kan sleutel uitlezen uit het exporteer bestand en deze omzetten naar de afbeelding (zie paragraaf 7.7), maar een programmeur kan geen afbeelding inladen door een statische sleutel mee te geven. Als dit wel wenselijk is zou er eventueel een 'tag' property kunnen worden toevoegt aan de elementen in de lijst. Deze zou dan wel handmatig moet worden ingesteld in de editor, wat een user interface zal vereisen.

Deze lijst zal tijdens de eerste keer opstarten van de editor opgebouwd en weggeschreven worden naar een JSON-bestand in de 'ProjectSettings' folder. Wanneer de lijst aanwezig is in de 'ProjectSetting' zal deze worden ingeladen tijdens het opstarten van de editor.

Om de lijst bij te houden terwijl de editor open is zal de 'assets' folder worden 'gewatched'. Dit houdt in dat de editor een event afvuurt wanneer er een bestand aangemaakt, bewerkt, verplaatst of verwijderd wordt. NodeJS beschikt over een module genaamd 'filesystem' die deze functionaliteit aanbied. Daarentegen blijkt deze functionaliteit van NodeJS vrij inconsistent is, dit meldt NodeJS ook in de documentatie[68]. Met twee events, zijnde 'change' en 'rename', is het lastig om te achterhalen wat precies het event afvuurde.

Er kan gebruik gemaakt worden van een library, zoals Chokidar[2], die meer consistentie biedt. Deze library biedt een interface met expliciete events zoals 'add, 'change' en 'unlink'. Hiernaast zijn er ook events voor directory beschikbaar zoals 'addDir' en 'unlickDir'. Chokidar wordt gebruikt in veel projecten[3], waaronder Microsoft's visual studio code[4].

7.5.1 Optimalisatie

Het refereren naar assets is een veel voorkomende actie in de editors. Wanneer de gebruiker in 'image content' naar een afbeelding wilt refereren zal de lijst gefilterd moeten worden op assets met het type 'image' (figuur 7.1). Het iedere keer moeten filteren van een potentiele grote lijst aan assets kan mogelijk, afhankelijk van de grootte van de lijst, zorgen voor een slechte gebruikerservaring. De tijdcomplexiteit van het algoritme in figuur 7.1 kan worden uitgedrukt als *O(n)*, waarin 'n' het aantal elementen in de array zijn.

[2] https://github.com/paulmillr/chokidar
[3] https://www.npmjs.com/browse/depended/chokidar
[4] https://github.com/microsoft/vscode

```
availableAssets.filter(asset => asset.type === 'image');
```

Figuur 7.1: Het filteren van assets met het type 'image'.

Om het vinden van assets met een bepaald type efficiënter te maken kan de editor een JavaScript object bijhouden die dient als een 'lookup table'. In dit object dienen de properties als keys die ieder een array van assets bevatten met dat type (zie figuur 7.2). De tijdcomplexiteit van een property lookup zoals in figuur 7.3 kan worden uitgedrukt als *O(1)*.

```
{
    images: [
        ..
    ],
    videos: [
        ..
    ]
}
```

Figuur 7.2: Geoptimaliseerd formaat voor het beheren van assets.

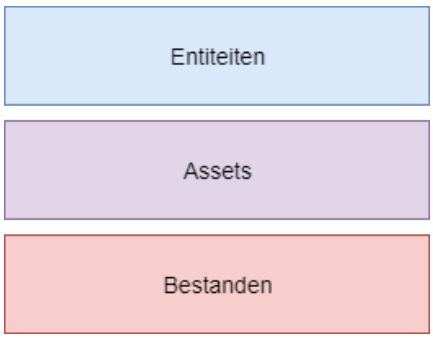
```
const availableImages = availableAssets.images;
```

Figuur 7.3: Het verkrijgen van alle beschikbare images door middel van een property lookup

7.6 Semantiek binden aan assets

Om de editors minder foutgevoelig te maken wordt er semantiek in de vorm van een type toegewezen aan de assets. Zo moeten de nieuwe editors alleen het gebruikt van afbeeldingen in 'image content' toelaten. Door een type toe te wijzen aan assets kunnen types worden afgedwongen in de editor.

In deze paragraaf is er sprake van twee semantische lagen boven die gebonden worden aan bestanden die beschikbaar zijn voor de editor: de asset- en entiteiten laag. De asset laag classificeert bestanden op bestandstypen zoals afbeelding of video. De entiteiten laag categoriseert en koppelt deze assets aan hogere constructen zoals karakters en locaties.



Figuur 7.4: Semantische lagen in game content.

Voor de eerste semantische laag, het hangen van types aan bestanden, moeten de volgende stappen worden ondernomen:

1. De nieuwe editor moet een type kunnen hangen aan bestanden wanneer deze ingeladen worden.
2. Het content typen dataschema moet typen kunnen
3. De inspector moet het afgedwongen typen in het dataschema respecteren en eventuele andere typen weigeren.

Boven op de assets laag wordt de entiteiten laag gebouwd.

4. Assets moeten gecategoriseerd en gekoppeld kunnen worden aan hogere constructen.

7.6.1 Bestanden koppelen aan een type

Er kan een type aan een assets gekoppeld worden door de extensie van het bestand te respecteren. Zo is een '.png' of '.jpeg' een afbeeldingen en een 'mp4' een video. Deze semantische laag wordt vanaf nu de 'asset laag' genoemd. De nieuwe editor zal in code over een map moeten beschikken waarin een bestand extensie gekoppeld staat aan een type. Alleen extensies die ondersteund worden door het NGT zullen moeten worden opgenomen in deze map. Deze map wordt gedefinieerd in broncode, omdat deze wordt geacht om statisch te zijn. Als de map aangepast moet worden zal de programmeur dit moeten doen, omdat hier verdere consequenties aanhangen. Het toevoegen van een nieuwe mapping zal invloed hebben op het NGT, deze moet de nieuwe extensie ondersteunen. Hiernaast zal het omzetten van een mapping moeten resulteren in een migratie van de interne lijst aan beheerde assets.

7.6.2 Het uitbreiden van JSON-schema functionaliteit

Het content typen dataschema moet uitgebreid worden om informatie over het type asset te verstrekken. Refereren naar assets gebeurd door middel van een sleutel; de 'id' waarde van de elementen in de lijst. Een sleutel is een string wat betekend dat het JSON-schema voor een referentie property nog steeds 'string' waarde voor het 'type' keyword bevat. Er moet een keyword toegevoegd worden om metadata te verstrekken over het type asset.

JSON-schema verstrekt zelf al een 'format' keyword die semantische validatie toelaat. Een voorbeeld van een formaat is: 'email'[60]. De waardes van format mogen uitgebreid worden naar benodigde formaten. Maar omdat een 'id' niet in een onderscheidbaar formaat staat is het beter om een nieuw keyword toe te voegen waarin het type asset omschreven kan worden. Hiervoor wordt er een nieuw keyword geïntroduceerd: 'assetType'. Voorbeelden van waardes van dit keyword zijn: 'image', 'video', 'sound', ect ...

7.6.3 Het afdwingen van typen in de inspector

Om JSON-schema's te reflecteren in de inspector wordt gebruik gemaakt van de 'react-jsonschema-form' library, zoals vermeld in hoofdstuk 5. Het toevoegen van een nieuw keyword in het JSON-schema woordenboek betekend dat er nieuwe componenten gemaakt moeten worden die dit keyword ondersteunen. Deze component moet getoond worden wanneer het 'assetType' keyword aanwezig is in het schema.

Het ondersteunen van dit nieuwe keyword heeft een grote impact op de library. Om het 'assetType' keyword te ondersteunen moet het standaard gedrag van de library worden aangepast; het standaard veld voor een string type moet worden overschreven. De component die het string veld overschrijft zal de juist component teruggeven voor het schema.

7.6.4 Het koppeling van assets aan entiteiten

Vaak worden afbeeldingen van karakters en locaties geordend in een folderstructuur. Dit duidt op een behoefte aan het categoriseren van assets. Met een tweede semantische laag worden asset van hetzelfde typen handmatig gebonden aan hogere entiteiten zoals karakters en locaties. Hiermee kan worden voorkomen dat er per ongeluk een verkeerde afbeelding bij een bepaald karater wordt gebruikt.

Omdat het refereren naar assets gaat via een menselijk onleesbare sleutel zal het handmatig binden van assets aan entiteiten moeten gebeuren door middel van een user interface in de editors. Deze user interface zal de gebruiker meer informatie geeft over de asset waarmee de gebruiker deze asset kan identificeren. Wanneer deze gekoppeld wordt aan een entiteit zal de editor de bijbehorende sleutel zoeken.

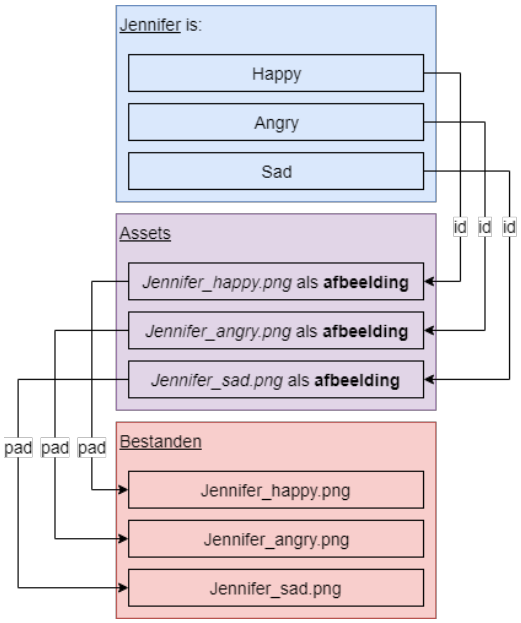
Deze semantische laag leeft niet in het content typen schema zoals de eerste semantische laag, maar op project niveau. Een geserialiseerd (zoals te zien in figuur 7.5) formaat zal dan ook worden opgeslagen in de 'ProjectSettings' folder. Het geserialiseerde formaat in figuur 7.5 is geoptimaliseerd zoals beschreven in alinea 7.5.1.

```
{
    "images": [
        {
            "jennifer": [
                "idAsset1",
                "idAsset2"
            ]
        }
    ],
    "videos": [
        ..
    ]
}
```

Figuur 7.5: Geserialiseerde representatie van de entiteiten laag.

Vervolgens zullen content typen met een referentie naar een asset een extra veld moeten tonen in de inspector. Dit extra veld is een drop down menu waarin een keuze gemaakt kan worden uit de eerder gecreëerde entiteiten. Vervolgens kunnen de assets gefilterd worden op de geselecteerde entiteit. Dit extra veld wordt niet gedefinieerd in het content typen schema en niet geëxporteerd. Het wordt wel opgeslagen in de save file die de editor produceert.

De relaties tussen de semantische lagen zijn terug te zien in figuur 7.6.



Figuur 7.6: Relaties tussen de verschillende semantische lagen.

7.7 Exporteer stap

Voorheen exporteerde de huidige editors slechts twee bestanden die het verhaal en gelokaliseerde inhoud omschreven. Een overkoepelende projectstructuur maakt het mogelijk om gebruikte assets, het verhaal en configuratie bestanden samen te bundelen in een archief format zoals tar[5] of asar[6]. Er is

[5] https://www.freebsd.org/cgi/man.cgi?query=tar&apropos=0&sektion=5&manpath=FreeBSD+7.0-RELEASE&arch=default&format=html

[6] https://github.com/electron/asar

niet gekeken naar een “best passende formaat” in deze context. Naast dat dit het laden van ongebruikte assets voorkomt verkleint dit het aantal HTTP requests, die gemaakt moeten worden om alle files binnen te halen. Een HTTP request bevat extra informatie welke iedere keer mee gestuurd zal worden[69]. Dit maakt het ophalen van één groter bestand sneller dan vele kleinere bestanden wat zorgt voor een snellere laadtijd van het spel.

Het NGT maakt gebruik van een library genaamd: ‘preloadJS’[7]. Met deze library worden bestanden ingeladen. De library biedt de mogelijkheid om een zogenaamde ‘manifest’ op te stellen. Dit is een JSON-bestand waarin paden naar bestanden worden gekoppeld aan een sleutel. De nieuwe editor kan een manifest specificeren, omdat deze beschikt over de sleutels en paden van gebruikte assets. Paden naar assets worden gestript van (sub)directories die zich bevinden in het pad, zodat alleen de bestandsnaam met extensie overblijft. Vervolgens wordt het basis pad gespecificeerd door de export locatie te pakken en de locatie van het NGT project hiervan af te trekken. Dit resulteert in een relatief pad dat het NGT leidt naar de locatie van de assets. Een voorbeeld van een manifest is terug te zien in figuur 7.7. Omdat dit proces zo alleen zou werken voor de ‘preloadJS’ library is het aan te raden om voor het genereren van een manifest gebruik te maken van het ‘strategy design pattern’, om zo ondersteuning voor eventuele andere libraries te bieden.

```
{
    "path": "assets/",
    "manifest": [{
            "id": "8e3c642d-ab1e-4c81-a379-0b236bf692f8",
            "src": "jennifer_happy.png"
        },
        {
            "id": "23951903-0f4e-43d2-9602-5fd6e70b0d32",
            "src": "jennifer_angry.png"
        }
    ]
}
```

Figuur 7.7: Voorbeeld van een manifest in preloadJS.

7.8 Conclusie

Met deze conclusie wordt deelvraag “Hoe kan game content in de overkoepelende projectstructuur verbonden en geordend worden?“ beantwoord.

Het gebruiken van een overkoepelende projectstructuur in de nieuwe editors bevorderd efficiëntie, functionaliteit en verminderd foutgevoeligheid. De folderstructuur van het NGT zal aangepast moeten worden om deze overkoepelende projectstructuur te ondersteunen.

Om game content te ordenen en met elkaar te verbinden in een overkoepelende projectstructuur zal de editor assets, bestanden die samen de game content opbouwen, moeten beheren. Dit wil zeggen dat de nieuwe editor een lijst met beschikbare assets zal bijhouden. Elementen in deze lijst bestaan uit een gegenereerde “menselijk onleesbare” unieke sleutel, een pad naar het bestand en het type bestand (e.g. afbeelding) die semantiek toekend aan de asset. De sleutel maakt het verbinden van assets mogelijk. Deze sleutel wordt gebruikt om te refereren naar de bijbehorende asset en zorgt dat referenties niet breken wanneer de asset verplaatst wordt. Een nadeel aan een menselijk onleesbare sleutel is dat alleen de editor kan refereren naar deze assets. De programmeur kan moeilijk refereren naar deze assets vanuit statische context. Bij iedere verandering in de lijst zal deze worden

[7] https://www.createjs.com/preloadjs

geserialiseerd en weggeschreven naar een bestand in de overkoepelde project folder, zodat deze uitgelezen kan worden wanneer de editor het project opent. Als dit bestand wordt verwijderd zullen alle referenties binnen de editors breken.

Semantiek wordt toegekend aan de assets door (1) automatisch een type te binden aan assets door middel van de bestandsextensie en (2) een handmatige stap waarbij assets aan entiteiten gebonden worden. Een voorbeeld van een entiteit is een karakter welke bestaat uit verschillende afbeeldingen die emoties representeren.

Wanneer de nieuwe editor het verhaal exporteert zullen alle gebruikte assets, configuratie bestanden en het verhaal samen gebundeld worden in een archief format zoals tar. Dit verminderd het aantal HTTP requests dat het NGT hoeft te maken om assets in te laden. Het NGT gebruikt de library 'preloadJS' om bestanden in te laden. Deze heeft een manifest nodig die opgesteld moet worden tijdens de exporteer stap.

7.9 Vervolgonderzoek

7.9.1 User interfaces voor beheren van de entiteiten laag

Het binden van assets aan entiteiten is een handmatig proces waarbij de gebruiker via een user interface assets moet kunnen koppelen aan (zelf aangemaakte) entiteiten. Hiervoor dient de gebruiker over een user interface te beschikken waarbij de gebruiker assets kan koppelen aan entiteiten, omdat het lastig is om te werken met menselijk onleesbare asset sleutels. Er is geen onderzoek gedaan naar hoe deze user interface met werken en eruit moet zien. Dit lijkt een behoefte voor een verkenner in de editors op te roepen die gebruikt kan worden om bestand te inspecteren. De user interface voor de koppeling tussen assets en entiteiten zou eventueel op dezelfde plek kunnen komen als de inspector, wanneer er een bestand geïnspecteerd wordt.

7.9.2 Exporteer formaat

De overkoepelende projectstructuur maakt het mogelijk om gebruikte assets, configuraties en het verhaal te archiveren in een formaat zoals tar. Er zou nog onderzoek gedaan moeten worden naar verschillende archief formaten en hun voor- en nadelen. Vervolgens moet er gekeken worden naar hoe het NGT deze kan uitpakken/ uitlezen.

Hoofdstuk 8

Conclusies

Hoofdstuk 9

Discussie

Hoofdstuk 10

Reflectie

Niet beschikbaar in concept versie.

Hoofdstuk 11

Referenties

[1] Anaïs Gibert, Wade C Tozer, and Mark Westoby. Teamwork, Soft Skills, and Research Training. *Cell press*, 2017.

[2] Barton J. Hirsch. Wanted: Soft skills for today's jobs. *Phi Delta Kappan*, 98, 2017.

[3] Marcel M Robles. Executive Perceptions of the Top 10 Soft Skills Needed in Today's Workplace. *Business Communication Quarterly*, 75(4):453–465, 2012.

[4] Ashutosh Bishnu Murti. Why Soft Skills Matter. *IUP Journal of Soft Skills*, 8(3), 2014.

[5] Bernd Schulz. The Importance of Soft Skills: Education beyond academic knowledge. *NAWA Journal of Language and Communication*, 2008.

[6] David Pizzi and Marc Cavazza. From Debugging to Authoring: Adapting Productivity Tools to Narrative Content Description. 2008.

[7] Chong-U Lim, Robin Baumgarten, and Simon Colton. Evolving Behaviour Trees for the Commercial Game DEFCON. 2010.

[8] Chris Johnson. Basic Research Skills in Computing Science. 2007.

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

[10] Open Source & SaaS Tools — StackShare.

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

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

[13] Apache Flex®.

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

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

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

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

[18] Apache Flex® - Download Apache FlexJS.

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

[20] Haxe - The Cross-platform Toolkit.

[21] Haxe Summit 2018 in Seattle.

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

[23] Trending Haxe repositories on GitHub this month.

[24] Haxe - The Cross-platform Toolkit.

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

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

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

[28] Stack Overflow Developer Survey 2018.

[29] About Github.

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

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

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

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

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

[35] Language Bindings - Qt Wiki.

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

[37] Home — Qt Forum.

[38] Diagram Scene Example — Qt Widgets 5.11.

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

[40] Qt - Valve Developer Community.

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

[42] Qt Interface - VideoLAN Wiki.

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

[44] Qt WebKit - Qt Wiki.

[45] Cutelyst.

[46] Wt, C++ Web Toolkit — Emweb.

[47] Electron Apps — Electron.

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

[49] 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.

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

[51] 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.

[52] Oscar Nierstrasz, Laurent Dami, Vicki de Mey, Marc Stadelmann, Dennis Tsichritzis, and Jan Vitek. Visual Scripting - Towards Interactive Construction of Object-Oriented Applications. 1991.

[53] tools-visual-scripting - Asset Store.

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

[55] 3D Visual Scripting · inexorgame/inexor-core Wiki.

[56] 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.

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

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

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

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

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

[62] A Wright and H Andrews. JSON Schema: A Media Type for Describing JSON Documents, 2018.

[63] T.A. Galyean III. Narrative Guidance of Interactivity. (1988):186, 1995.

[64] Thijs Schipper. *Doorontwikkeling OZ platform*. Master thesis, TU Delft, 2015.

[65] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[66] Abstract Factory Design Pattern.

[67] Builder Design Pattern.

[68] File System — Node.js v10.3.0 Documentation.

[69] R Fielding, UC Irvine, and J. Gettys. Hypertext Transfer Protocol – HTTP/1.1, 1999.

Appendices

Bijlage A

Assets.json

Bijlage B

scaffold2