HBO-ICT

Technisch Ontwerp

Low Code Editor | Aryzon.World

Versie: 3.1  
Vertrouwelijk: Ja  
Bedrijf: Aryzon.World  
Opleiding: HBO-ICT voltijd  
Uitstroomprofiel: SE, 3e jaar  
Bedrijfsmentors: Maarten Slaa, Henk Stomphorst  
Schoolbegeleider: Henk Bosman  
Studenten: Koen van Meijeren, Patrick Slag,

Tim Scholten en Pascal Meijerman

Afbeelding met Kleurrijkheid, licht, kunst

Automatisch gegenereerde beschrijving

**Team**

|  |  |  |  |
| --- | --- | --- | --- |
| Naam | E-mail | Student nummer | Functie |
| Henk Stomphorst | henk@aryzon.com | - | Product Owner |
| Maarten Slaa | - | - | Technische begeleider |
| Nicholas Wilkinson | - | - | Technische begeleider |
| Henk Bosman | h.bosman@windesheim.nl | - | Schoolbegeleider |
| Patrick Slag | patrick.slag@windesheim.nl | S1159962 | Developer |
| Tim Scholten | tim.scholten@windesheim.nl | S1158692 | Developer |
| Koen van Meijeren | koen.van.meijeren@windesheim.nl | S1153353 | Developer |
| Pascal Meijerman | pascal.meijerman@windesheim.nl | S1143521 | Developer |

**Versiebeheer**

|  |  |  |
| --- | --- | --- |
| Versie | Datum | Omschrijving |
| 0.1 | 09-05-2023 | Level 1 en 2 uitgewerkt |
| 0.2 | 12-05-2023 | Level 3 en delen van level 4 uitgewerkt |
| 0.3 | 17-05-2023 | Technisch keuzes beschreven |
| 1.0 | 06-06-2023 | C4-model opnieuw uitgewerkt en level 4 verder uitgewerkt |
| 3.0 | 20-06-2023 | Up-to-date gemaakt met resultaat van sprint 3 |
| 3.1 | 21-06-2023 | Structuur geoptimaliseerd en discussiepunten opgelost |

**Distributie**

|  |  |  |  |
| --- | --- | --- | --- |
| Naam | Rol | Datum | Versie |
| Henk Stomphorst | Bedrijfsbegeleider | 20-06-2023 | 3.1 |
| Maarten Slaa | Bedrijfsbegeleider | 20-06-2023 | 3.1 |
| Nicholas Wilkinson | Technische begeleider | 20-06-2023 | 3.1 |
| Henk Bosman | Schoolbegeleider | 20-06-2023 | 3.1 |

Inhoud

[1 Inleiding 4](#_Toc138239574)

[2 Aryzon.World-applicatie 5](#_Toc138239575)

[3 Oculus-platform 6](#_Toc138239576)

[4 Low Code Editor 8](#_Toc138239577)

[4.1 Interne werking Low Code Editor 9](#_Toc138239578)

[4.1.1 Components 9](#_Toc138239579)

[4.1.2 Event Menu 16](#_Toc138239580)

[4.1.3 Connection Maker 17](#_Toc138239581)

[4.1.4 Expressies 19](#_Toc138239582)

[4.1.5 Multiplayer 23](#_Toc138239583)

[5 Template System 26](#_Toc138239584)

[5.1 Interne werking Template System 27](#_Toc138239585)

[5.1.1 Template UI 27](#_Toc138239586)

[5.1.2 Template IO 27](#_Toc138239587)

[6 Toepassing van de SOLID-principes 29](#_Toc138239588)

[6.1 Single Responsibility 29](#_Toc138239589)

[6.2 Open-Closed 29](#_Toc138239590)

[6.3 Liskov Substitution 29](#_Toc138239591)

[6.4 Interface Segregation 29](#_Toc138239592)

[6.5 Dependency Inversion 29](#_Toc138239593)

[7 Bibliografie 30](#_Toc138239594)

# Inleiding

Dit ontwerp beschrijft de uitgangspunten voor de applicatie op technisch niveau. Dit document is gestructureerd volgens het C4-model. Het C4-model houdt in dat de applicatie steeds gedetailleerder beschreven wordt. Het model is opgedeeld in vier lagen: context, container, componenten en code.

In het eerste hoofdstuk wordt beschreven hoe het systeem op hoofd-niveau eruitziet. In het tweede hoofdstuk wordt ingezoomd op de applicatie waar het team aan gaat werken en worden de technische componenten op hoog niveau weergegeven. In de daaropvolgende hoofdstukken wordt ingezoomd op de containers waar het team aan gaat werken en worden de componenten van elke container in beeld gebracht. Tenslotte worden in laagste niveau de technische keuzes die zijn gemaakt beschreven. [1]

# Aryzon.World-applicatie

Aryzon.World is een applicatie die mensen de mogelijkheid geeft om virtuele werelden te maken en te spelen. Bij de applicatie horen twee actoren: de wereldbeheerder en de wereldspeler. De wereldbeheerder gebruikt Aryzon.World voor het bouwen van werelden. De wereldspeler gebruikt Aryzon.World om de werelden die gemaakt zijn te spelen.

Aryzon.World is opgebouwd uit verschillende platformen, namelijk Oculus, standalone en mobiel. Alle platformen maken gebruik van het Core softwaresysteem. Dit systeem is verantwoordelijk voor het verwerken van bewerkingen in de wereld en biedt generieke functionaliteiten aan. Het team werkt tijdens dit project voornamelijk aan het Oculus-platform.

Afbeelding met tekst, schermopname, diagram, Lettertype

Automatisch gegenereerde beschrijving

Figuur 1 – Aryzon.World-systeem overzicht

# Oculus-platform

Het Oculus-platform biedt de VR-applicatie aan. Deze applicatie bestaat uit meerdere containers, namelijk Authentication, World, Input System, Multiplayer, World editor, de nieuwe Low Code Editor en Template System. Alle containers maken gebruik van Core om wereldbewerkingen uit te kunnen voeren en gebruik te maken van generieke functionaliteiten.

Afbeelding met tekst, schermopname, diagram, Lettertype

Automatisch gegenereerde beschrijving

Figuur 2 – Oculus-platform containers

Na het opstarten van de applicatie, wordt de gebruiker gevraagd om zich te authentiseren, dit is het doel van de Authentication container. Hierna krijgt de gebruiker toegang tot de applicatie en worden de functionaliteiten voor de wereld aangeboden via de World container. Dit betreft onder andere het startscherm voor het laden of maken van een wereld en het gebruiken van de wereld.

Om ervoor te zorgen dat de wereld door meerdere actoren te gelijk gebruikt kan worden, is de Multiplayer container ontwikkeld. Deze container draagt zorg voor het synchroniseren van de wereld naar alle andere actoren. Tijdens het werken met de wereld en het gebruiken van de applicatie wordt er gebruik gemaakt van input van de gebruiker. Om dit goed te kunnen verwerken en volledig gebruik te kunnen maken van de features die de VR-bril biedt, wordt het Input systeem gebruikt.

Tenslotte zijn de laatste containers de Low Code Editor en het Template System. Beide containers zijn nieuw en het team gaat hieraan werken. Beide containers worden als onderdeel van de World Editor container ontwikkeld. De Low Code Editor biedt het nieuwe systeem aan voor het koppelen van events aan wereldobjecten. Het Template System zorgt ervoor dat de huidige staat van wereldobjecten weggeschreven kan worden in een template. Op deze manier kunnen wereldobjecten eenvoudig opnieuw gebruikt worden in een andere of dezelfde wereld. Hiervoor maakt het Template System gebruik van het bestandssysteem.

# Low Code Editor

De Low Code Editor container bestaat uit meerdere onderdelen. De basis hiervan ligt in het components onderdeel. Dit onderdeel bevat alle type components, zoals de trigger en property components, en biedt de logica aan voor het bouwen en configureren van een flow. Op basis van de gemaakte flow kan dan tenslotte een event gemaakt en gekoppeld worden aan een wereldobject.

Afbeelding met tekst, schermopname, diagram, Lettertype

Automatisch gegenereerde beschrijving

Figuur 3 - Componenten van de Low Code Editor container

Het Event Menu component is verantwoordelijk voor het aanbieden van het menu voor het plaatsen van de components in de wereld. De Connection Maker biedt de functionaliteiten aan voor het maken van flow tussen de componenten. Als de componenten volledig geconfigureerd zijn en de flow is af, dan kan de flow gebouwd worden. Tijdens het bouwen van de flow wordt de flow omgezet naar expressiedata en kan tenslotte het event gemaakt en gekoppeld worden aan een wereldobject. Hierbij wordt gebruik gemaakt van de Expressions component, dit component is een schil om het bestaande expressiedatasysteem dat gemaakt is door Aryzon.World. Dit systeem is echter erg complex en daarom biedt dit component een vereenvoudiging aan voor het werken met expressiedata.

Om ervoor te zorgen dat de configuratie van de blokken in multiplayer beschikbaar is, wordt er gebruik gemaakt van de multiplayer component. Verder worden ook de connecties tussen componenten via dit systeem gesynchroniseerd. Het is belangrijk om code zoveel mogelijk te kunnen hergebruiken, daarom is de Util component ontwikkeld. Dit component is verantwoordelijk voor het aanbieden van, veelal statische, classes en methoden waarin herbruikbare code aangeboden wordt. Denk hierbij aan de factories voor het werken met expressiedata en utility voor het uitlezen van resources. Dit component wordt verder niet uitgelegd in level 4, omdat de meester helper classes al beschreven zijn in de uitleg van de andere componenten.

Tenslotte maakt de Low Code editor gebruik van het Core softwaresysteem. De componenten van de Low Code Editor container maken hier veelvuldig gebruik van, denk hierbij aan het ophalen van de wereldobjecten, het synchroniseren van data in multiplayer en het koppelen van een event aan een wereldobject. Hierdoor kan het team in theorie bestaande code makkelijk hergebruiken, in de praktijk was dit vaak wat lastiger. Zie ook het adviesrapport over de Aryzon.World codebase voor de problemen waar het team tegen aan liep.

## Interne werking Low Code Editor

In dit hoofdstuk wordt verder ingegaan op de structuur van de componenten van de Low Code editor. Per onderdeel wordt uitgelegd hoe het onderdeel in elkaar zit en welke technische keuzes zijn gemaakt. De volgorde is gebaseerd op de volgorde van gebruik in de applicatie.

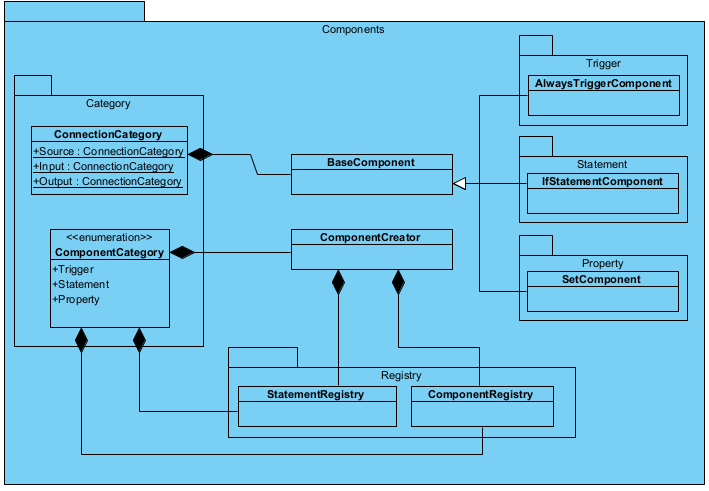
### Components

De basis van het project zijn de bouwblokken, die intern ook wel components worden genoemd. Een component heeft een categorie, een type en één of meerdere connectiepunten. Deze componenten worden opgeslagen in registries, en worden in de wereld aangemaakt door de ComponentCreator. Verder zijn de componenten alleen zichtbaar wanneer de wereld in edit modus is. Dit is ontworpen op basis van gesprekken met de opdrachtgever en is al vroeg in het project uitgedacht.

Afbeelding met diagram

Automatisch gegenereerde beschrijving

Figuur 5 - Eerste design van de structuur voor de blokken en flow



Figuur 6 - Components package structuur

Op hoog niveau is dit de structuur van de components package. In de volgende subhoofdstukken wordt dit verder uitgelegd en beschreven. Met behulp van deze components kan er in de wereld een visuele flow gebouwd, welke gebruikt kan worden om een event te bouwen en te koppelen aan een object.

#### Component Category

Het startpunt van het systeem zijn de component categorieën. Elk component valt onder één van de volgende categorieën: trigger, statement of property. In de volgende subhoofdstukken wordt elke categorie uitgelegd.

##### Trigger

Een trigger is degene die de hele flow in gang zet. Een trigger kan een conditie hebben, zoals ‘als object X met object Y collide doe dan een actie’, maar het kan ook altijd runnen. De trigger is dus degene die bepaald wanneer het event wordt uitgevoerd.

##### Statement

Een statement is een breed begrip. Met statements wordt hier types als selection statements en iteration statements bedoeld. Dus bijvoorbeeld een if statement, switch statement en for statement. Op zichzelf doet een statement dus niet veel, maar het wordt gebruikt voor het bouwen van een event en zorgt ervoor dat je condities kan stellen aan een event, maar op een andere manier dan bij triggers.

##### Property

Onder property valt alles dat gerelateerd is aan values. Denk aan het getten van een property van een object en het setten van een property op een ander object.

#### Component Type

Onder component types vallen alle individuele components van de component categorieën, zoals de if statement voor statements en de object collision trigger van triggers.

#### Connectiepunten

De connectiepunten zijn de witte bolletjes die half in een component zitten. Aan de linkerkant zitten de source connecties, aan de bovenkant zitten de input connecties en aan de rechterkant de output connecties. De connectiepunten worden intern ook wel connectables genoemd. Met behulp van deze connectiepunten kan gecommuniceerd worden tussen de verschillende components. Elk connectiepunt heeft een specifiek taak, de source

##### Source connecties

De source connectie is het startpunt van een component. Alleen triggers hebben geen source connectie, omdat de triggers het begin van de flow zijn. De source connectie zorgt ervoor dat een component uitgevoerd kan worden, dat wil zeggen dat de flow omgebouwd kan worden naar expressiedata. Een voorbeeld van hoe source connecties worden toegepast is in een if-statement die verbonden is met een trigger. Hierbij is de trigger de source voor de if-statement en is de if-statement verbonden met de output van de trigger. Een belangrijk punt om rekening mee te houden is dat de source connectie ook een return statement is. Dit wordt bijvoorbeeld gebruikt in de get property component. De waarde van de property is de return value. Deze return value kunnen andere componenten zoals de set property weer gebruiken. Bij meerdere source connecties is de eerste connectie onderin en de laatste connectie bovenin.

A picture containing text, screenshot, design

Description automatically generated

Figuur 7 - De waarde van de ‘get property’-blok is tegelijk de input van de ‘set property’-blok

##### Input connecties

De input connecties zijn connecties die een output waarde van een andere connectie ontvangen en daarmee iets kunnen doen. Bij meerdere input connecties is de eerste connectie links en de laatste connectie rechts.

##### Output connecties

De output connecties zijn connecties die andere componenten conditioneel kunnen aanroepen. Bij meerdere output connecties is de eerste connectie bovenin en de laatste connectie onderin. In het geval van een trigger blok wordt de output altijd geretourneerd, maar in het geval van een if-statement hangt dat af van of de conditie waar is.

#### Technische samenhang

De trigger componenten zijn verantwoordelijk voor het omzetten van de flow in ExpressionData. Deze data worden gebruikt voor het creëren van EventData, waarmee het event gekoppeld worden aan een wereldobject. Hiervoor wordt gebruik gemaakt van bestaande code van Aryzon.World. Elk trigger component bevat een ‘BuildFlow’-methode. Deze methode voert de hierboven logica uit, verwijderd de oude events en koppelt het nieuwe event aan het wereldobject.

Verder heeft BaseComponent een private readonly Dictionary genaamd “\_connections” die als key ConnectionCategory heeft en als value “List<List<BaseComponent>>”. Dit lijkt qua structuur erg op “connectables” die als key ConnectionCategory heeft en als value “List<GameObject>”. Dit komt omdat het ook zo is, dat het erg op elkaar lijkt. Het enige verschil is dat één connectable meerdere connections kan hebben. Bijvoorbeeld de if-statement. De if-statement heeft één connectable voor true en één voor false. Maar de true connectable kan meerdere components aanroepen (en dus heeft één connectable meerdere connections.) Dit is de reden dat het een “List<List<BaseComponent>>” is in plaats van “List<BaseComponent>”.

#### Component Registry

De ComponentRegistry is verantwoordelijk voor het registreren van alle objecten die gebruikt kunnen worden voor het bouwen van een flow. Deze is verder opgedeeld in één registry per component category. Hierdoor kunnen er makkelijk uitbreidingen gedaan worden op de structuur en is het event menu modulair gebouwd. In onderstaande afbeelding wordt de relatie tussen de registry en de components afgebeeld.

Afbeelding met tekst, diagram, schermopname, Plan

Automatisch gegenereerde beschrijving

Figuur 8 - Codestructuur registries

##### Toevoegen van een component

Om een component toe te kunnen voegen moeten er 3 dingen gebeuren. De eerste stap is het maken van een klasse voor de component die BaseComponent implementeert. Vervolgens moet er een prefab gemaakt worden voor de component waarbij het gemaakte script toegevoegd moet worden. De prefab moet geplaatst worden onder het mapje “/Resources/LowCode/\*component category\*/”. Verder moet de prefab een collider hebben zodat het kan berekenen waar de connectables geplaatst moeten worden en heeft het een Unity component met de naam Title nodig, waar het programma de naam van de component in kan zetten (bijvoorbeeld “If” voor IfStatementComponent.)

Om een component toe te voegen hoeft er alleen een field gemaakt te worden in de component type zijn registry waarin de locatie naar de prefab staat (vanaf “/Resources/LowCode/\*component category\*/”), de label die weergegeven wordt in het event menu en een beschrijving voor in het uitleg menu.

##### Toevoegen van een component category

Om rekening te houden met de uitbreidbaarheid van dit systeem, is de code modulair opgebouwd. Het is eenvoudig om nieuwe categorieën (bijvoorbeeld operators) en daarvoor componenten toe te voegen. De eerste stap is het aanmaken van een nieuwe class genaamd “\*component category\*Registry” in de namespace “LowCode.Components.Registry”. Hieronder is een voorbeeld van een simpele Component Category Registry.A screen shot of a computer program

Description automatically generated with low confidence

Figuur 9 - Voorbeeld van hoe een registry werkt

De nieuwe klasse moet een property genaamd Components hebben, want de volgende stap is om aan de klasse ComponentRegistry het volgende toe te voegen aan zijn “InitComponents”: “components.AddRange(\*component category\*Registry.Components.Value);”. De volgorde van die regel is verantwoordelijk voor de volgorde waarin de Components worden weergegeven in het event menu.

##### Het gebruik van Lazy

Een van de dingen die waarschijnlijk opvallen als er naar de registries gekeken wordt, is het gebruik van Lazy. Dit was eigenlijk een overblijfsel van een eerdere versie van de registries (toen er nog geen gebruik gemaakt werd van het Core softwaresysteem), maar nadat de blokken in multiplayer werd gesynchroniseerd, bleek dat dit niet meer kon. Dit komt omdat er in een latere versie ook local variables worden gebruikt in de ComponentCreator. Deze variabelen moeten pas tijdens het initialiseren van het blok worden aangemaakt. Doordat er gebruik gemaakt wordt van Lazy, wordt het blok pas aangemaakt als er om gevraagd wordt.

#### Component Creator

Voor het aanmaken van components in de wereld wordt de ComponentCreator gebruikt. Deze creator zorgt ervoor dat er een ObjectStructure aangemaakt wordt voor de component. De ObjectStructure wordt gebruikt om het object aan te maken via multiplayer. Het bevat eigenschappen zoals de naam, positie en hoe een object eruitziet. Ook wordt in de component creator de local variable gelinkt aan het object. De component creator wordt gebruikt door het event menu. Het menu geeft dan ook aan wat van soort type component gemaakt moet worden.

### Event Menu

Het event menu is een lijst die over erft van de ItemDisplayList class die meerdere EventBlockListItems bevat die weer over erven van DisplayListItem. De ItemDisplayList en DisplayListItem zijn abstracte classes die in het event menu en in het templatesysteem worden gebruikt.

Afbeelding met tekst, schermopname, diagram, lijn

Automatisch gegenereerde beschrijving

Figuur 10 - Event menu package structuur

Het Event menu is het menu waaruit de component blokken geselecteerd kunnen worden. De naam van het menu is het event menu aangezien dit de naam is dat het oude systeem had, maar intern worden blokken componenten genoemd. Deze blokken kunnen aan de wereld toegevoegd worden om een flow te bouwen. De objecten worden uit de ComponentRegistry gehaald. De werking van de componenten en registries zijn in hoofdstuk 4.1.1 besproken.

Afbeelding met tekst, schermopname, visitekaartje

Automatisch gegenereerde beschrijving

Figuur 11 - Voorbeeld van het Event Menu in Aryzon.World

Voor het weergeven van de objecten is er een EventBlockList gemaakt, deze loopt door alle objecten in de componentRegistry en vult de lijst met EventBlockListItems. De EventBlockListItems krijgen een ComponentCreator mee die helpt om de blokken te instantiëren als er op een EventBlockListItem wordt gedrukt.

### Connection Maker

Elk bouwblok dat aangemaakt wordt via het menu bevat een vooraf ingestelde componenten structuur. In deze structuur is de 'ConnectionMaker' als component opgenomen. Een van de functionaliteiten die hierdoor aangeboden kan worden is het herkennen van klik-events. Aan de main-scene is het 'ConnectionController' component toegevoegd. Dit component is verantwoordelijk voor het onthouden van de aangeklikte objecten. En aanmaken van lijnen. Zie onderstaande afbeelding voor de structuur.

Afbeelding met tekst, schermopname, diagram, lijn

Automatisch gegenereerde beschrijving

Figuur 12 - ConnectionMaker klassendiagram

Voor het maken van een connectie tussen twee objecten moet de wereldbeheerder de objecten aanklikken. Het eerste object wat aangeklikt wordt de 'Origin', het tweede object de ‘Destination’. Als zowel de origin als de destination ingesteld zijn, dan wordt er een LineHolder aangemaakt waarin een lijn tussen de bouwblokken wordt gerenderd. Ook wordt deze connectie opgeslagen om de flow van events te kunnen maken. In onderstaande sequence diagram wordt getoond hoe dit request verloopt. Er is gekozen voor deze aanpak omdat op deze manier het eenvoudig is om connecties te maken tussen objecten. Het beperkt het aantal handelingen wat een gebruiker moet doen om een connectie te maken en het zorgt ervoor dat connecties makkelijk bewerkt kunnen. Alle connecties samen creëren uiteindelijk de flow waarmee het event gebouwd kan worden.

Afbeelding met tekst, diagram, Parallel, schermopname

Automatisch gegenereerde beschrijving

Figuur 13 - Voorbeeld van hoe een request verloopt voor het maken van een connectie tussen components

### Expressies

Om de bouwblokken om te zetten naar echte functionaliteit in de wereld wordt het omgezet naar expressies. Deze expressies worden dan gedeeld over multiplayer en kunnen dan tijdens playmode uitgevoerd worden.

#### Expressiedata

Om functionaliteit toe te voegen aan objecten tijdens het spelen van de game wordt gebruik gemaakt van events. Een events bestaat uit een condition en een response. Deze conditie en response worden opgebouwd uit expressie data. De expressiedata van de condition leidt tot een boolean. Wanneer deze boolean een true teruggeeft wordt de respone van het event afgevoerd. De expressie data van de response wordt gebruikt om properties van gameobjecten aan te passen.

Omdat het best complex kan zijn om de expressiedata te genereren, is ervoor gekozen om gebruik te maken van factories voor. Zie onderstaande afbeelding voor een voorbeeld van een helper method voor het genereren van expressiedata.

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

Figuur 14 - Voorbeeld van het maken van expressiedata

#### Expressiedata voor triggers

Om een event te starten moet de trigger, oftewel de conditie van het event, true retourneren. Deze expressie data voor trigger wordt gegenereerd door een factory class. De verschillende triggers gebruiken verschillende methodes in de factory class om ervoor te zorgen dat ze de trigger specifieke expressie data krijgen. De expressiedata die de triggers hiervan krijgen wordt gebruikt om de flow te starten.

Afbeelding met tekst, schermopname, lijn, diagram

Automatisch gegenereerde beschrijving

Figuur 15 - Expressiedata ophalen voor de trigger van een event

#### Expressiedata voor properties

Om een property van een object aan te kunnen passen of op te halen, moet er expressiedata gegenereerd worden die als response van een event afgevoerd kan worden. Deze code wordt gegeneerd door een factory class. Een property blok kan nu op basis van de geselecteerde property een methode aan roepen in de factory class die vervolgens de expressiedata retourneert die uitgevoerd kan worden als een response van een event. Een groot voordeel van deze class is dat dit het gebruiken van expressiedata veel eenvoudiger maakt. Het genereren van de expressiedata werkt voor de property components op eenzelfde manier als voor de trigger components.

#### Expressiedata voor het bouwen van de flow

De expressiedata die hierboven is uitgelegd wordt aangeroepen wanneer een wereldbeheerder op het “Build” icoon klikt van een trigger. De build knop roept ‘BuildFlow’-methode aan die de ‘Execute’-functie van de trigger aanroept, die de ‘Execute’-functie van alle verbonden componenten aan roept die weer de ‘Execute’-functie van al hun verbonden componenten aanroept en zo verder. In figuur 13 wordt afgebeeld hoe de flow gemaakt kan worden met behulp van de blokken. Intern wordt deze flow omgezet naar expressiedata en gekoppeld aan een wereldobject. In figuur 14 wordt de interne werking van het bouwen van de expressiedata voor het maken van een event voor dezelfde flow weergegeven.

Afbeelding met schermopname

Automatisch gegenereerde beschrijving

Figuur 16 - Voorbeeld van hoe de flow in de wereld gemaakt wordt

Afbeelding met tekst, diagram, lijn, schermopname

Automatisch gegenereerde beschrijving

Figuur 17 - Voorbeeld van hoe de gebouwde flow wordt omgezet naar expressiedata

### Multiplayer

Zoals hierboven uitgelegd is, bestaat het nieuwe systeem uit verschillende type components welke aan de wereld toegevoegd kunnen worden. Met behulp van de Connection Maker kan vervolgens een flow gemaakt worden en kan de flow omgezet worden naar expressiedata om zo een event te kunnen bouwen. Maar deze configuratie wordt nog niet in multiplayer gesynchroniseerd, hiervoor is de multiplayer component verantwoordelijk. Dit component zorgt ervoor dat data gesynchroniseerd wordt in multiplayer en regelt het serializeren en de-serializeren van de data. De uiteindelijk afhandeling hiervan ligt bij Aryzon.World.

Afbeelding met tekst, schermopname, diagram, Parallel

Automatisch gegenereerde beschrijving

Figuur 18 - Multiplayer component structuur

#### Local variables managers

Alle componenten krijgen tijdens het toevoegen van het object aan de wereld een aantal voor gedefinieerde variabelen toegewezen. Tijdens het configureren van de blokken worden deze variabelen bijgewerkt. Hiervoor wordt gebruik gemaakt van de local variables managers. Deze managers bieden de variabelen aan voor de configuratie van een component en zorgen ervoor dat de waarde van de variabelen bijgewerkt en uitgelezen kunnen worden. Hiervoor wordt gebruik gemaakt van de multiplayer data helper.

#### Multiplayer data helper

De multiplayer data helper regelt het aanmaken en bijwerken van variabelen. Omdat de Aryzon.World Multiplayer component best complex is, is ervoor gekozen om een helper class te maken die dit systeem vereenvoudigd. Deze helper biedt functies aan voor het aanmaken, uitlezen en bijwerken van variabelen. Toch blijft het werken met variabelen lastig, doordat er van tevoren een variabele moet worden gedeclareerd op een hard-coded index, waarmee vervolgens later de variabele bijgewerkt kan worden.

#### Component connecties opslaan in multiplayer

Daarnaast is er ook nog de beperking dat alle waarden geconverteerd moeten worden naar een string. In de meeste gevallen is het relatief eenvoudig om de data te converteren, int naar string en visa versa. Maar in het geval de connecties is dit wat moeilijker, hierbij moet een dictionary bestaande uit connectie categorieën en een lijst van componenten geconverteerd worden naar een string. Bij het uitlezen moet vervolgens de string weer omgezet worden naar een dictionary met dezelfde data als voor het parsen. De uitdaging hierbij was dat in dictionary niet-unieke data stonden, maar voor het converteren van de data moest dit wel unieke-data zijn. Op basis van deze data moest in een nieuwe sessie weer dezelfde connecties opgebouwd kunnen worden. Uiteindelijk is er gekozen voor een implementatie waarbij het unieke object ID, de connectie category en de connecties op te slaan, waarbij de connecties op eenzelfde manier waren opgebouwd, namelijk: unieke object ID, connectie category en de index van de connectable. In onderstaande afbeeldingen wordt een voorbeeld gegeven van de data die opgeslagen wordt als string.

Afbeelding met schermopname

Automatisch gegenereerde beschrijving

Figuur 19 - Voorbeeld flow waarvan de data opgeslagen wordt in multiplayer

De connectie data voor het If-statement-blok levert na het converteren de output van onderstaande afbeelding op. In onderstaande afbeelding heeft het If-statement-blok geen verbinding met de player collision trigger, is er een verbinding vanaf de source connectie met true-value-blok en is er verbinding vanaf de eerste output connectie naar de eerste input connectie van het set-property-blok.

Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijving

Figuur 20 - Voorbeeld van hoe de connecties worden opgeslagen in multiplayer

Het systeem voor het converteren van de connecties en andere waarden is niet perfect, maar dit is zo tot stand gekomen doordat het systeem waarmee gewerkt wordt niet goed is getest door Aryzon.World. Bovendien werd er in dit project voor het eerst met dit systeem gewerkt. Hierdoor zaten er nog een aantal fouten in, zoals het aanmaken van variabelen op basis van hard-coded getallen. Het was bijvoorbeeld niet mogelijk om een waarde op te slaan op basis van datatype, maar moest dit geconverteerd worden naar een string.

# Template System

Dit systeem bestaat uit twee componenten, namelijk Template UI en Template IO. De Template UI component heeft als doel om alle front-end van de templates af te handelen. De Template IO component heeft als doel om de backend aan te bieden voor het templatesysteem. Daarbij wordt gebruik gemaakt van het bestandssysteem voor het aanmaken en uitlezen van bestanden. Het Core softwaresysteem wordt gebruikt voor het opslaan van geselecteerde objecten en het omzetten van een template naar wereldobjecten.

Afbeelding met tekst, schermopname, Lettertype, diagram

Automatisch gegenereerde beschrijving

Figuur 4 - Componenten van de Template System container

## Interne werking Template System

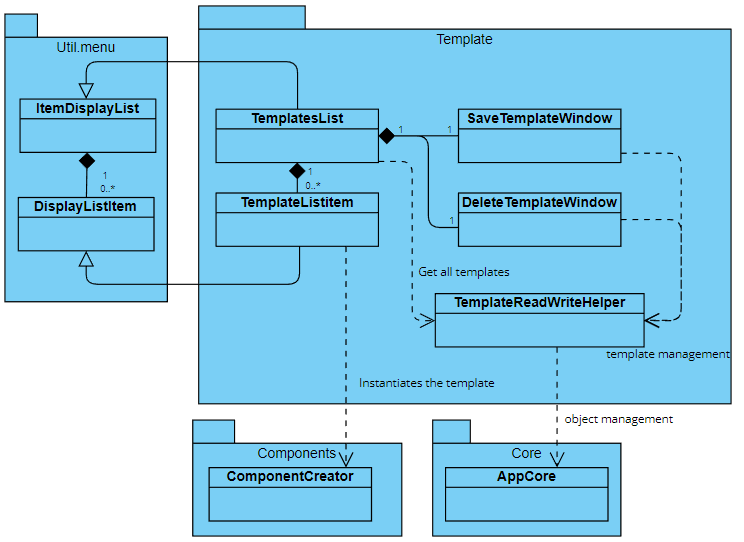
Het Template System zorgt ervoor dat de huidige staat van wereldobjecten weggeschreven kan worden in een template en zo opnieuw gebruikt kan worden in een andere of dezelfde wereld. Hiervoor maakt het Template System gebruik van het bestandssysteem. In dit hoofdstuk wordt de structuur besproken en de technische keuzes toegelicht.

### Template UI

De Template UI heeft drie voorname doelen namelijk: het tonen van alle gemaakte templates, een menu tonen voor het maken van templates en een menu tonen voor het verwijderen van templates. Het tonen van alle gemaakte templates wordt gedaan doormiddel van TemplateListItems elk van deze items bevat één template, deze worden getoond op dezelfde manier als het Event block menu.

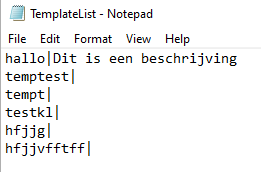
### Template IO

De Template IO heeft als doel om alles wat te maken heeft met het maken, laden en verwijderen van templates. Dit wordt allemaal gedaan door de TemplateReadWriteHelper, deze class kan nieuwe templates aanmaken en opslaan in een tekstbestand. Verder kan het ook alle aangemaakte templates ophalen uit tekstbestanden en omzetten naar bruikbare data en wordt het verwijderen van de templates aangeboden.



Figuur 21 - Structuur van het templatesysteem

De TemplateReadWriteHelper gebruikt tekstbestanden voor het opslaan van de templates. Een van deze bestanden is de opslag voor alle gemaakte templates, en heeft de naam TemplateList.txt. In dit bestand worden de namen en beschrijvingen van de templates opgeslagen met als scheidingsteken een '|'. Dit is gedaan om bij te houden welke templates er zijn. Om een template op te slaan wordt er in een tekstbestand één object per regel opgeslagen. Op de laatste regel staat het middelpunt van de template in de vorm van een 3D vector, deze wordt gebruikt om de positie van de template op de correcte plek te plaatsen. Hieronder is een voorbeeld te zien van de inhoud van het template list bestand.



Figuur 22 - Template list bestandsstructuur

# Toepassing van de SOLID-principes

In dit hoofdstuk wordt besproken hoe de verschillende SOLID principes [2] zijn toegepast.

## Single Responsibility

De klassen die gemaakt zijn voor dit project voldoen aan single responsibility principe doordat ze zich richten op één doel. Een goed voorbeeld hiervan zijn de trigger components. Deze classes zijn zo opgebouwd, dat alleen wanneer de werking van het blok moet veranderen, de klasse aangepast hoeft te worden. Alle andere verantwoordelijkheden liggen buiten de class, zoals het renderen van het blok of het opbouwen van de expressiedata voor het blok. Voor het ophalen van expressiedata wordt gebruikt gemaakt van een factory, welke als doel heeft om expressiedata te genereren. Zo heeft elke class maar één verantwoordelijkheid en dus één reden om te veranderen. In de code is op meerdere plekken eenzelfde soort aanpak gebruikt, meestal in de vorm van helper klassen.

## Open-Closed

Er wordt voldaan aan het Open-Closed principe in de gemaakte classes, omdat classes openstaan voor uitbreiding, maar gesloten zijn voor aanpassingen. Dit is bijvoorbeeld goed te zien bij de components, de multiplayer data helper of de play mode listener. Deze classes zijn zo gebouwd dat het uitbreiden van de functionaliteit eenvoudig is, maar dat de code niet aangepast hoeft te worden om eropin te kunnen haken.

## Liskov Substitution

Dit principe is gevolgd bij de implementatie van de components. De implementatie komt hiermee overeen doordat er geen components zijn die niet vervangen kunnen worden door een ander component-type. Elk component-type heeft dezelfde vereiste functies voor het genereren van expressiedata en levert hetzelfde resultaat op. Op deze manier is dit principe gevolgd.

## Interface Segregation

Niet van toepassing.

## Dependency Inversion

Hoewel de code niet gebruik gemaakt heeft van dependency injection, is dit principe toch op een aantal plekken gevolgd. Wanneer er bijvoorbeeld interactie was met de components, dan verliep dit zoveel mogelijk via de abstracte class in plaats van concrete implementaties.

# Bibliografie

|  |  |
| --- | --- |
| [1] | [Online]. Available: https://c4model.com/. |
| [2] | Y. K. Erinç, „freecodecamp,” [Online]. Available: https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/. |