Afstudeerverslag

Skies of Saturn

Danny van den Bree

2014

|  |  |
| --- | --- |
| http://www.fourcelabs.com/wp-content/themes/FourceLabs/images/logo-fourcelabs.png | E:\Dropbox\Sail - Skies of Saturn 2014\Art\Indigo GFX\SoS logo transparant.png |

Titel: Het opzetten van een eco-systeem (kunstmatige intelligentie) binnen het spel Skies of Saturn.

Student: D.F. van den Bree

Studentnummer: 08052026

Onderwijsinstelling: De Haagse Hogeschool

Opleiding: Informatica

Aftstudeerperiode: 10-02-2014 t/m 06-06-2014

Examinator: dhr. Mijnarends & dhr. Schouten.

Bedrijf: FourceLabs

Adres: Neude 5, Utrecht

Bedrijfsleider: Karel Millenaar

Datum 22-9-2014

Auteur: Danny van den Bree

Versie: 2.0

Versiebeheer

|  |  |  |  |
| --- | --- | --- | --- |
| Versie | Datum | Eigenaar | Aanpassingen |
| 0.1 | 20-2-2014 | Danny van den Bree | Eerste versie |
| 0.2 | - | Danny van den Bree | Sprints toegevoegd |
| 0.3 | - | Danny van den Bree | Planning |
| 0.4 | - | Danny van den Bree | Bouwen ecosysteem |
| 0.5 | - | Danny van den Bree | Inhoudsopgave |
| 0.6 | - | Danny van den Bree | Voorwoord |
| 0.7 | - | Danny van den Bree | Testen |
| 2.0 | 22-9-2014 | Danny van den Bree | Reparatie versie 2 |

Voorwoord

Ik zou graag een aantal collega’s willen bedanken die betrokken zijn geweest bij mijn afstuderen.

* Karel Millenaar, voor de kans die hij mij heeft geboden voor het uitvoeren van deze opdracht.
* Justin van Luijk, voor de begeleiding tijdens mijn afstudeerstage.
* Mijn overige collega’s bij FourceLabs, voor de geweldige werksfeer.

“Before software can be reusable, it first has to be usable.”

**- Ralph Johnson (computer scientist)**

Samenvatting

De afstudeerstage van Danny van den Bree heeft plaatsgevonden bij FourceLabs, een gameontwikkelaar in Utrecht. Eén van de hoofdproducten van FourceLabs is de Wip'n'Kip. Wip'n'Kip is in 2010 ontwikkeld als een PLAY Pilots live game, mogelijk gemaakt door de Provincie Utrecht en Gemeente Utrecht. Wip'n'Kip is een live game met volwassen formaat wipkippen, bedoeld voor festivals en evenementen. Drie spelers strijden tegen elkaar door fysiek zo hard mogelijk te wippen, terwijl zij een virtuele kip tot de eindstreep brengen.

FourceLabs houdt zich niet alleen bezig met fysieke games maar ook digitale games voor iPhone en iPad met onder andere Skies of Saturn. Skies of Saturn heeft als doel om de speler een relaxte speelervaring te bieden doormiddel van abstracte art en geluid. Je kunt Skies of Saturn vergelijken met één van de volgende games; Lumini[[1]](#footnote-1), Flower[[2]](#footnote-2) of Journey[[3]](#footnote-3).

Gedurende het afstuderen is het project verdeeld in verschillende onderdelen. In het eerste deel wordt de opdracht, de organisatie en het Plan van Aanpak beschreven. In het tweede deel worden de opgeleverde producten en de beslissingen die zijn genomen tijdens de ontwikkeling behandeld. In het laatste deel de evaluatie van het projectproces en de opgeleverde producten. Met het toelichten van de toepassing van competentie sluit ik dit afstudeerverslag af.

Ten slotte is het volledige afstudeerproject gedocumenteerd in deze scriptie. Aan het eind van de afstudeerperiode is het ecosysteem dat is ontwikkeld opgeleverd. Alle geplande hoofddoelen van het project zijn afgerond. Dit zorgt ervoor dat het project succesvol is afgerond.

Verklarende Woordenlijst

|  |  |
| --- | --- |
| **Term** | **Omschrijving** |
| Skies of Saturn | De game in kwestie. |
| LibGDX | LibGDX is een 2D rendering engine is een game-development applicatie framework geschreven in Java. Het geeft de ontwikkelaar de mogelijkheid om games te ontwikkelen voor desktop en mobile met dezelfde stuk code. LibGDX is Multi platform dat op PC, Mac, Linux, Android, IOS en webbrowsers met webGL support. |
| Spine | Spine is een pakket speciaal voor LibGDX ontwikkeld waarmee je animaties kunt maken. |
| Java | Java is een programmeertaal. |
| Eclipse | Een Java editor. |
| A\* | Een pathfinding algoritme dat de snelste route berekend tussen twee punten. |
| SCRUM | Een iteratieve agile ontwikkelingsmethode, waar korte sprints gebruikt  worden om producten te ontwikkelen. |
| Scrum meeting | Dagelijkse bijeenkomsten van het hele team om voortgang in kaart te brengen en obstakels duidelijk te maken. |
| Product Backlog | Een lijst met functieomschrijvingen van een product. |
| Sprint | Binnen SCRUM is een sprint een korte ontwikkelingsperiode. Hierin wordt  een product ontwikkelt en opgeleverd. |
| Sprint Planning | Een planning wordt gemaakt voor een nieuwe sprint, en bepaald de inhoud van de Sprint Backlog. |
| Sprint Backlog | Hierin staat tot in detail beschreven wat de functies zijn die aan het einde van de Sprint af moeten zijn. |
| Usecase | In een use case is gewenste functionaliteit beschreven. Vervolgens kunnen  tests worden uitgevoerd die controleren of het voldoet aan de eisen. |
| Klasse | Binnen object-georiënteerde programmeertalen is een klasse een blauwdruk voor objecten. Het voorziet de startwaardes en de  implementatie van het gedrag |
| MoSCoW | Een methodiek om prioriteit aan eisen te geven. De prioriteiten lopen van  hoog naar laag: Must, Should, Could en Would. |
| Object | In object-georiënteerde programmeertalen is een object een instantie dat  waardes en functionaliteit kunnen bevatten. |
| DGG | Dutch Game Garden |
| Shippable code | Het doel van iedere Sprint: bruikbare, geteste functies afleveren voor het product. |
| Pathfinding | De kortste route berekenen tussen twee punten. |

Inhoudsopgave

[Inleiding 1](#_Toc400351577)

[De projectomgeving 2](#_Toc400351578)

[De organisatie 2](#_Toc400351579)

[Collega’s 3](#_Toc400351580)

[Plaats van de afstudeerder binnen FourceLabs 3](#_Toc400351581)

[Bedrijfsbegeleider 3](#_Toc400351582)

[Rolverdeling 4](#_Toc400351583)

[Opdrachtformulering 5](#_Toc400351584)

[Aanleiding opdracht 5](#_Toc400351585)

[Algemene beschrijving Skies of Saturn 5](#_Toc400351586)

[Probleemstelling 5](#_Toc400351587)

[Doelstelling 5](#_Toc400351588)

[Opdrachtformulering 6](#_Toc400351589)

[Randvoorwaarden 6](#_Toc400351590)

[Risicoanalyse 6](#_Toc400351591)

[Op te leveren producten 6](#_Toc400351592)

[Methodes, technieken en software 7](#_Toc400351593)

[Game Engine 7](#_Toc400351594)

[Programmeertaal 7](#_Toc400351595)

[Modelleertaal 7](#_Toc400351596)

[IDE 7](#_Toc400351597)

[Ontwikkelmethode 8](#_Toc400351598)

[Uitvoering van de afstudeeropdracht 9](#_Toc400351599)

[Backlog en followup sprints 9](#_Toc400351600)

[De sprints met daarbij hun activiteiten 10](#_Toc400351601)

[Sprint 1 – Vooronderzoek (10 februari tot 1 maart) 11](#_Toc400351602)

[Sprint 2 – Opzet framework (1 maart tot 7 maart) 14](#_Toc400351603)

[Sprint 3 - Physics en world generation (7 maart tot 14 maart) 19](#_Toc400351604)

[Sprint 4 - Player mechanics (14 maart tot 1 april) 25](#_Toc400351605)

[Sprint 5 - front end en Tree generation (1 april tot 7 april) 31](#_Toc400351606)

[Sprint 6 - Post processing & Parallax background (7 april tot 14 april) 35](#_Toc400351607)

[Sprint 8 - Defining AI types & Parallax background (14 april tot 1 mei) 38](#_Toc400351608)

[Sprint 9 – Path finding (1 mei tot 20 mei) 42](#_Toc400351609)

[Evaluatie 44](#_Toc400351610)

[Beantwoording doelstellingen 44](#_Toc400351611)

[Evaluatie opgeleverde producten 45](#_Toc400351612)

[Te behalen competenties 46](#_Toc400351613)

[In het vervolg beter of anders zou doen 47](#_Toc400351614)

[Bijlage A 48](#_Toc400351615)

[Het spel 48](#_Toc400351616)

[Gameplay 49](#_Toc400351617)

[Vliegen 49](#_Toc400351618)

[Mobiele platforms 49](#_Toc400351619)

[Muisbesturing 49](#_Toc400351620)

[Gamepad besturing 49](#_Toc400351621)

[Het basis 50](#_Toc400351622)

[De wereld van Skies 52](#_Toc400351623)

[Windmarkers 54](#_Toc400351624)

[Glyphs 55](#_Toc400351625)

[Glyphs oppakken 55](#_Toc400351626)

[Glyphs gebruiken 55](#_Toc400351627)

[Souvenirs 55](#_Toc400351628)

[Bijlage B 56](#_Toc400351629)

[Tileset 56](#_Toc400351630)

[Physics 56](#_Toc400351631)

[Noisemap 58](#_Toc400351632)

[Wind particles 59](#_Toc400351633)

[Flyer animation 60](#_Toc400351634)

[Tree generation 61](#_Toc400351635)

[Parallax Background 62](#_Toc400351636)

[Vignette Shader 63](#_Toc400351637)

# Inleiding

Voor u ligt mijn afstudeerverslag van mijn afstudeerstage bij FourceLabs van februari tot en met juni 2014. Tijdens mijn afstudeerperiode bij FourceLabs heb ik gewerkt aan de ontwikkeling van kunstmatige intelligentie voor het spel Skies of Saturn. Het doel van dit verslag is inzage geven in het proces dat gevolgd is zodat er bepaald kan worden dat alle vooraf gedefinieerde competenties zijn behaald.

Het onderwerp van de afstudeeropdracht is het toevoegen van een ecosysteem aan het spel Skies of Saturn. Skies of Saturn is een spel waar exploratie vooraan staat in een abstracte wereld. Dit spel kun je vergelijken met de games Lumini, Flower of Journey. De ontwikkeling wordt verzorgd door FourceLabs in samenwerking met DGG-genoten (Dutch Game Garden) Gray Lake Studios & Hence Games.

Dit verslag is opgedeeld in drie delen. In het eerste deel wordt de opdracht, de organisatie en het Plan van Aanpak beschreven. In het tweede deel worden de opgeleverde producten en de beslissingen die zijn genomen tijdens de ontwikkeling behandeld. In het laatste deel de evaluatie van het projectproces en de opgeleverde producten. Met het toelichten van de toepassing van competentie sluit ik dit afstudeerverslag af.

# De projectomgeving

Dit hoofdstuk zal er nader gekeken worden naar het stagebedrijf FourceLabs waar ik de gedurende afstudeerperiode heb gewerkt. Eerst wat voor soort bedrijf FourceLabs is, in welk gebied zij werkzaam zijn en wat voor soorten diensten zij leveren. Daarnaast wie mij persoonlijk heeft begeleid en mijn plaats binnen FourceLabs.

## De organisatie

FourceLabs is gevestigd in het centrum van Utrecht, bestaat uit vijf vaste werknemers en een aantal stagiairs die zich bezig houden met de ontwikkeling van games voor recreatie en educatieve omgevingen. FourceLabs ontwerpt fysieke spellen en speelse omgevingen die een brug vormen tussen de spelers en de wereld van de klant. Deze games spelen zich af in de publieke ruimte. Door speelse ervaringen aan te bieden vragen en houden we de aandacht. FourceLabs ontwerpt onder meer voor events, musea en cross-media campagnes.

FourceLabs is eigenlijk sinds enige tijd al ‘FourceLab’. Van de oorspronkelijk vier oprichters (four) is er nog slechts één over, Karel Millenaar. Deze designer & artist houdt zich vooral bezig met het zoeken naar uitdagingen waaruit waanzinnige en leuke projecten uit voorkomen. Uiteraard zoekt en vindt hij daarbij de nodige hulp. In het geval van zijn nieuwe game, Skies of Saturn, is dat DGG[[4]](#footnote-4)-genoten Gray Lake Studios[[5]](#footnote-5) & Hence Games[[6]](#footnote-6).

FourceLabs ontwikkeld games voor onder andere:

* Festivals, marketing campagnes (Partijen die graag spektakel willen en de aandacht willen vestigen op hun boodschap of product)
* Wachtruimtes, winkelcentra (Publieke ruimtes die ervaring arm zijn, of graag en wat positievere uitstraling zouden hebben)
* Musea, educatieve instellingen (Organisaties die een inhoud wil ontsluiten op een manier die beklijft)

## Collega’s

Binnen FourceLabs draaien verschillende projecten die totaal een andere richting opgaan. Van fysieke games tot digitale games op groot scherm in een publieke ruimte. Het FourceLabs team bestaat uit vijf hoofdpersonen waardoor er geen zelfstandige teams aanwezig zijn. In plaats van afgebakende teams zijn we verdeelt over meerdere projecten en zorgt vooral voor meer diversiteit.

|  |  |
| --- | --- |
| Werknemers FourceLabs | Rol binnen FourceLabs |
| Karel Millenaar | “President of Play” Eigenaar (Artist) |
| Justin Luijk | “Overseer” Project Manager |
| Kenneth Nelson | “Super Designer” Game Designer |
| Marc de Vreede | “Firestarter” Designer |
| Danny van den Bree | “Master of Code” Programmer |
| Kitty Calis | Marketing (freelancer) |

|  |  |
| --- | --- |
| Overige personen | Werkzaamheden |
| Paul Veer | Pixel Artist, werkt onder andere aan de game Nuclear Throne (huurt ruimte, niet in dienst van FourceLabs) |

## Plaats van de afstudeerder binnen FourceLabs

Ik, de afstudeerder is bezig geweest met de ontwikkeling van kunstmatige intelligentie voor het spel Skies of Saturn. Om deze opdracht juist te kunnen uitvoeren ben ik geplaatst tussen twee teams: FourceLabs en Hence Games. Deze twee teams hebben mij geholpen met het bedenken van concept ideeën voor het ecosysteem.

## Bedrijfsbegeleider

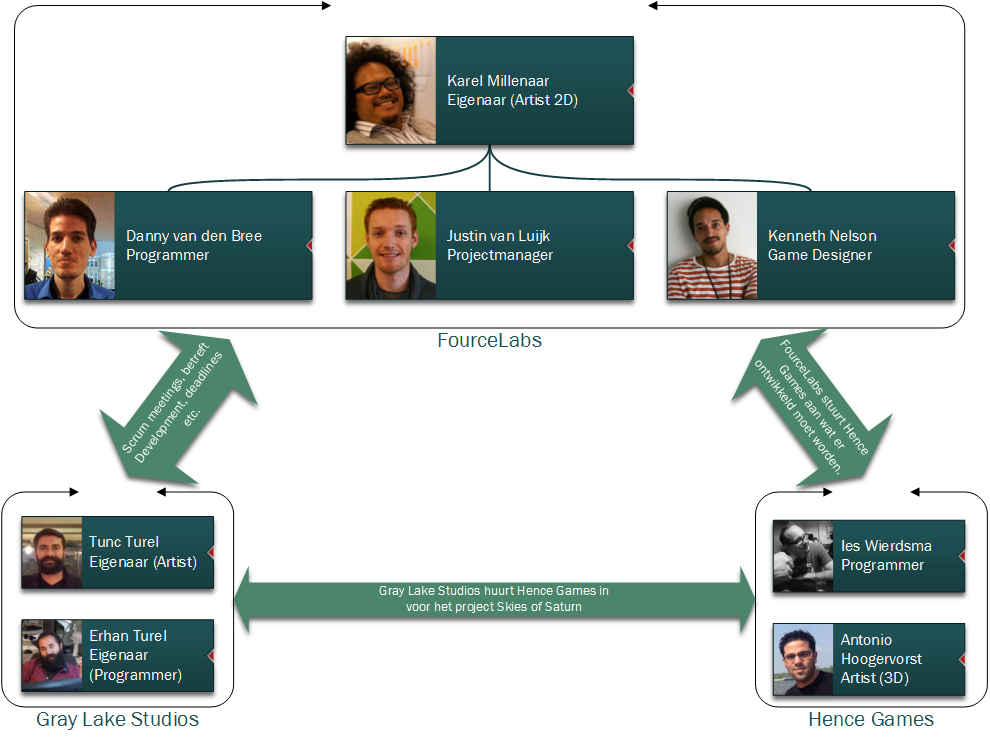
In mijn afstudeerperiode ben ik begeleid door Justin van Luijk, de projectmanager van FourceLabs. Justin houdt zich bezig met het plannen van projecten, zorgt dat projecten de deadlines halen en verzorgt de communicatie tussen verschillende partijen.

## Rolverdeling

Vanuit FourceLabs zijn er vier personen bij het project betrokken, de opdrachtgever (Eigenaar) Karel Millenaar, Projectmanager Justin van Luijk, Kenneth Nelson (Game Designer) en de opdrachtnemer: Danny van den Bree. Naast FourceLabs zijn er twee andere partijen betrokken bij het project Skies of Saturn; namelijk Gray Lake Studios en Hence Games. Gray Lake Studios is in grote lijnen partner van FourceLabs en regelt met Hence Games de development. Hence Games doet het modelwerk (3D) en grotendeels het programmeerwerk voor Skies of Saturn.

Het project is gefinancierd door Gamefonds[[7]](#footnote-7), zij zijn een partij dat eist dat degene die een subsidie aanvraagt het plan met een volledig team kan uitvoeren. FourceLabs en Gray Lake Studios hebben gezamenlijk de aanvraag ingediend en met deze constructie zijn de volgende rollen als volgt ingedeeld.

* FourceLabs: 2D artist, Projectmanager en game designer. (later kwam ik erbij als programmeur)
* Gray Lake Studios: programmeur
* Hence Games: Programmeur en artist.



Figuur 1

# Opdrachtformulering

## Aanleiding opdracht

FourceLabs is een project per 1 januari 2014 gestart in samenwerking met Gray Lake Studios en Hence Games, genaamd Skies of Saturn. Voor het project Skies of Saturn is FourceLabs verantwoordelijk voor het deel gamedesign en art en Gray Lake Studios zowel Hence Games voor het programmeergedeelte. Hence Games heeft sinds de start van het project, de besturing van de speler en “procedural generation[[8]](#footnote-8)” in ontwikkeling gehad en vrij weinig gedaan betreft AI (kunstmatige intelligentie). Aanleiding van mijn afstudeeropdracht had FourceLabs een “ecosysteem” voor ogen. Dit ecosysteem krijgt de mogelijkheid om de interacties te verzorgen tussen verschillende entiteiten, deze entiteiten kunnen zijn: wezens, flora en fauna, artefacten en glyphs. In de bijlage Skies of Saturn wordt er dieper op de inhoud ingegaan. In het ecosysteem kun je als speler, entiteiten met verschillende kenmerken aantreffen. Deze entiteiten zullen allemaal hun eigen karakteristieken meekrijgen. Deze entiteiten krijgen naast hun gedragingen functies die noodzakelijk zijn, bijvoorbeeld pathfinding[[9]](#footnote-9), om een entiteit door een wereld met hindernissen zonder problemen te kunnen laten navigeren.

## Algemene beschrijving Skies of Saturn

Skies of Saturn is een 2D single player game waarin exploratie centraal staat. De game is een samenwerkingsverband tussen FourceLabs en Grey lake studios en wordt in eerste instantie ontwikkeld voor iOS en Android met eventuele toekomstige uitbreiding naar PC, Mac en console platformen. In het spel Skies of Saturn ga je als speler vliegend op de wind door de Skies of Saturn. Je gaat op ontdekking door een uitgestrekte wereld naar onbekende gebieden. Het spel gaat vooral om de rustgevende ervaring.

## Probleemstelling

Voor Skies of Saturn zijn er nog geen interacties tussen de speler en kunstmatige intelligentie gedefinieerd. Daarbij deze probleemstelling; Welke aspecten van en bij het spel Skies of Saturn zou moeten worden aangepast en-of toegevoegd om de gewenste beleving qua interacties binnen een abstracte wereld te bewerkstelligen? Karel Millenaar kwam tot een conclusie dat een ecosysteem dit probleem zou moeten verhelpen. Hence Games heeft op verschillende manieren dit proberen op te lossen door eenvoudige vijanden te plaatsen in de wereld, dit had alleen meerwaarde op visueel gebied. Doordat de wereld procedureel (bij elke speelsessie willekeurig) gegenereerd wordt, moest de gameplay anders aangekleed worden. Door willekeurig entiteiten te plaatsen in een wereld maakt het niet interessant genoeg om de speler vijf minuten te blijven fascineren (korte speelsessies als doel).

## Doelstelling

Door de speler een unieke beleving mee te geven kreeg ik van FourceLabs de opdracht een ecosysteem te ontwikkelen waarbij de wereld van Skies of Saturn levendiger en actiever zal maken. Zonder dit ecosysteem zal er weinig te beleven zijn in de wereld van Skies of Saturn behalve het kunnen genieten van het artwork en muziek. Doelstelling van de opdracht is, is het toevoegen van een interactiesysteem dat herhaling en verveling wegneemt. Dit interactie systeem verzorgt de interacties tussen entiteiten en de speler. Een type entiteit zou bijvoorbeeld kunnen zijn; flora en fauna. Entiteiten krijgen de mogelijkheid om zich anders te gedragen op basis van de elementen die in hun nabije omgeving te vinden zijn. Daarnaast dat deze entiteiten hun eigen doelen kunnen vaststellen, onverwachte beslissingen kunnen nemen en doormiddel van “pathfinding” hun pad kunnen uitstippelen.

## Opdrachtformulering

Er zal een ecosysteem in het spel Skies of Saturn moeten worden geïmplementeerd waar de speler de mogelijkheid krijgt om in dit ecosysteem interactie aan te gaan met verschillende entiteiten: wezens, flora en fauna, artefacten en glyphs. In het ecosysteem kun je als speler entiteiten met verschillende kenmerken aantreffen. Deze entiteiten zullen allemaal hun eigen karakteristieken meekrijgen. Deze entiteiten krijgen naast hun gedragingen functies die noodzakelijk zijn, bijvoorbeeld pathfinding, om een entiteit door een wereld met hindernissen zonder problemen te kunnen laten navigeren. Deze wezens en flora en fauna zijn entiteiten die de wereld een opvulling geeft en heeft verder geen extra betekenis. Waarin juist artefacten en glyphs een gevolg kunnen hebben, bepalend wat je als speler ermee gaat doen. Aan deze artefacten en glyphs worden vervolg interacties gekoppeld waarin je als speler bijvoorbeeld delen van de wereld vrij kan spelen of bepaalde entiteiten kan activeren.

## Randvoorwaarden

Om dit project te kunnen starten moet er een aantal randvoorwaarden worden vastgesteld. Al het onderzoek wordt door mij verricht en hoeft daardoor geen kosten gemaakt te worden omdat ik in dienst ben als stagiair. Ik heb minimaal één werkplek nodig om mijn werk uit te kunnen voeren en die werkplek wordt vrijgesteld op het kantoor van FourceLabs. Ik werk op mijn eigen laptop, deze voldoet aan de eisen om prototypes mee te kunnen bouwen. Voor expertise moet ik zijn bij collega’s van Hence Games omdat dit bedrijf met de technische kant bezig zijn van Skies of Saturn. Er wordt geen groot onderzoek verricht, er worden vooronderzoeken verricht indien nodig en hebben betrekking tot bepaalde functionaliteiten.

## Risicoanalyse

Aan dit project zitten geen tot weinig risico’s aan verbonden, dit omdat als het ecosysteem uiteindelijk niet in het eind product wordt verwerkt eventueel andere oplossingen voor worden bedacht. Eén van de oplossingen zou zijn om het visueel te verwerken en de speler de illusie te geven dat het interactie model intelligent is. Daarom zou eventuele risico zijn als er geen interactie model wordt geïmplementeerd dat het spel fundamenteel iets mist. Door dit te vermijden ga ik het interactie model zo laagdrempelig mogelijk maken waardoor het team van Hence Games zonder problemen dit kan implementeren.

## Op te leveren producten

Er zal aan het einde van afstudeerperiode een aantal producten worden opgeleverd. Via deze producten kan ik vervolgens aantonen wat ik gedaan heb tijdens mijn afstudeerstage.

Deze producten zullen als volgt zijn:

* Oplevering ecosysteem (code)
* Plan van aanpak
* Procesverslag

# Methodes, technieken en software

## Game Engine

Toen ik net begon met mijn afstudeerstage stonden er vele evenementen op de planning waardoor het AI gedeelte laag op de prioriteitenlijst kwam te staan. FourceLabs wilde graag een speelbare demo laten zien aan klanten, dit betekende dus dat er op visual gebied, features gemaakt moesten worden. Dit werdt overigens verwerkt door Hence Games. Omdat de focus lag op iPhone en iPad gebruikers moest er rekening gehouden worden met de rekenkracht van deze mobiele apparaten. Doordat er twee bedrijven werkten aan Skies of Saturn was mijn rol niet helemaal duidelijk, code niet beschikbaar, moest via omwegen mijn weg zien te vinden. Hence Games ontwikkeld Skies of Saturn in Unity en ik heb gekozen om de ontwikkeling voort te zetten in LibGDX[[10]](#footnote-10) omdat ik zelf voorafgaand mijn afstudeerstage een aantal games met deze engine had gebouwd. LibGDX is een Java multiplatform 2d engine dat gespecialiseerd is in development van 2D games. Met LibGDX kon ik individueel aan Skies of Saturn werken zonder interventies van FourceLabs en Hence Games die aan de commerciële Unity build werkten.

## Programmeertaal

Java is een objectgeoriënteerde programmeertaal. Java is een platformonafhankelijke taal die qua syntaxis grotendeels gebaseerd is op de (eveneens objectgeoriënteerde) programmeertaal C++. Java beschikt echter over een uitgebreidere klassenbibliotheek dan C++. De LibGDX engine is volledig in Java geschreven waardoor ik automatisch met deze taal ben meegegaan.

## Modelleertaal

Het opzetten van een robuust structuur voor het ecosysteem kun je UML[[11]](#footnote-11) gebruiken. Voordat er gebouwd gaat worden kun je met UML kun je diagrammen moduleren zien hoe bepaalde onderdelen met elkaar gaan communiceren. UML zal vooral gebruikt gaan worden om de abstractie van verschillende entiteiten te kunnen definiëren.

## IDE

Eclipse is een opensource framework van de Eclipse Foundation voor software ontwikkelomgevingen. De bekendste toepassing is het gebruik als IDE voor de programmeertaal Java. De Eclipse omgeving is een zeer uitgebreide editor. Ik heb voor de Android Eclipse build[[12]](#footnote-12) gekozen zodat ik direct aan de slag kon met Android geïnstalleerde apparaten.

## Ontwikkelmethode

Er zijn een aantal ontwikkelmethodes beschikbaar voor software development waaronder waterval[[13]](#footnote-13), RUP[[14]](#footnote-14), Prince2[[15]](#footnote-15) en Scrum[[16]](#footnote-16). Ik heb voor Scrum gekozen omdat het bedrijf FourceLabs deze methode voor het project Skies of Saturn al hanteerde. Scrum is ontwikkeld om in korte zogeheten Sprints van tussen de zeven en vijftien dagen een gebruiksklaar stuk software af te leveren. Prioriteiten en doelen worden tijdens een “sprint planning” meeting bepaald, en komen uit een “Product Backlog”[[17]](#footnote-17). Zie het als een soort MoSCoW model om te bepalen wat minimaal een vereiste is om tot een volwaardig product te komen.

Het team van FourceLabs bepaald aan de hand van de Product Backlog wat er voor de komende sprint ontwikkeld gaat worden. Dit is belangrijk, omdat het einddoel niet meer gewijzigd dient te worden tot aan het einde van de Sprint. Voordat de eerste sprint afgetrapt kon worden moest er product backlog gedefinieerd worden met alle functionaliteiten die wij als FourceLabs graag wilden zien in het ecosysteem. Een backlog is een lijst met features die je graag in jouw project zou willen implementeren, ook de features die niet noodzakelijk zijn. Wekelijks werd er een meeting gehouden voor wat er in de aankomende week op de planning stond. Een meeting wordt gehouden tussen mijzelf en Justin van Luijk. Justin is de projectmanager en begeleid onder andere het project Skies of Saturn.

Scrum kan op verschillende manieren uitgevoerd worden, in ons geval maakte wij gebruik van het Trello systeem. Trello is een board waarop je kunt aangeven hoelang een sprint gaat lopen en het verwachtte resultaat. Dit werd in mijn gehele afstudeerperiode bijgehouden. Op Trello maakte ik een aantal work cards aan, een voorbeeld hiervan is, “speler controller”. Per work card wordt er aangegeven; de prioriteit en eventueel een sub list waarin je een opdracht in kleinere opdrachten opbreekt. Trello board is als volgt opgedeeld; Features (long term), To Do (this week), Doing (for today), Iterate, Waiting for confirmation en als laatst Done. Op deze board hebben Justin en Karel toegang toe, zij beslissen uiteindelijk zodra ik een work card naar “Waiting for confirmation” sleep, gecontroleerd wordt of deze daadwerkelijk uitgevoerd was.

Uitvoering van de afstudeeropdracht

## Backlog en followup sprints

Alle functionaliteiten die ik heb opgesteld voor mijn project en die FourceLabs graag terug ziet in het eindproduct.

|  |  |
| --- | --- |
| Begin framework voor prototyping | Cut scènes |
| Splash Screen | Main Menu (dynamic background) |
| Credits | Menu Settings |
| Physics gebaseerde verlichting | Flyer (de speler) textures, trail en animaties |
| Tile Painting | UI Test platform |
| Post processing shaders: Vignetten + Sepia | ABL |
| Xbox controller | Post processing shader: Background |
| Entity system (DoD) | Level vervormingen |
| Ambient music + gameplay audio | Onderzoek naar path finding |
| A Star algorithm (path finding) |  |
|  |  |
| WorldMap (programmeren) | WorldManager (programmeren) |
| PhysicsManager (programmeren) | WorldCamera (programmeren) |
| Misc (programmeren) | AssetManager (programmeren) |
| SceneManager (programmeren) | SoundManager (programmeren) |
|  |  |
| Noise map generatie | WorldMap bitwise tile selectie |
| Parallax background | Het maken van verschillende civilisatie scenario’s |
| Tree generatie, tree spawning | Brainstorming verschillende entiteiten |
| Besturing speler | Boids |
| Wind particles (Windrichting, de speler krijgt snelheid op basis van de windrichting) | Apocalypse |
| Definiëren van civilisaties | Speler -> death -> recycle -> spawn Tree chain |

## De sprints met daarbij hun activiteiten

In dit onderstaande tabel staan alle sprints die zijn plaatsgevonden tijdens mij afstuderen.

|  |  |
| --- | --- |
| Sprint nr. | Features |
| Sprint 1 (Vooronderzoek) | Definiëren van civilisaties  Het maken van verschillende civilisatie scenario’s |
| Sprint 2 (Opzet framework) | Begin framework voor prototyping  Noise map generatie |
| Sprint 3 (Physics en world generation) | WorldMap class  WorldMap bitwise tile selectie  Misc class  Physics manager class  Physics gebaseerde verlichting  Flyer (de speler) textures, trail en animaties |
| Sprint 4 (Player mechanics) | Besturing speler  World Manager class  Wind particles  Windrichting, de speler krijgt snelheid op basis van de windrichting  WorldCamera class |
| Sprint 5 (front end en Tree generation) | Tree generation  Spawn Tree  Speler -> death -> recycle -> spawn Tree chain  UI Test platform |
| Sprint 6 (Post processing) | Post processing shaders: Vignetten, Sepia en Background. |
| Sprint 7 (AI behaviors) | ABL[[18]](#footnote-18) (a behavior language) |
| Sprint 8 (Defining AI types & Parallax background) | Brainstorming verschillende entiteiten  Parallax background |
| Sprint 9 (Pathfinding) | Spawning Boids  Onderzoek naar pathfinding  A Star algoritme (path finding)  Level deformaties |

#### Sprint 1 – Vooronderzoek (10 februari tot 1 maart)

In de eerste week heb ik allerlei informatie verkregen van FourceLabs over het spel Skies of Saturn en liet me een aantal van hun demo’s spelen. Mijn eerste indruk was dat het spel puur om exploratie ging, dus ik vloog wat rond en gebeurde vrij weinig naast dat ik af en toe een paar “wezens” tegenkwam die je volgden en verder niets. Uiteindelijk ging je na ongeveer twee minuten spelen dood en dat was het einde. Conclusie, interactie was nogal karig. Doelstelling van het spel was dat je als speler moest crashen zodat daar “boompjes” uit konden groeiden. Leuk idee, alleen het werkt averechts omdat je juist alles ging ontwijken want je denkt dat, dit de juiste manier van spelen is.

De zaken die van toepassing zijn gekomen voor deze sprint zijn als volgt:

* Het uittekenen van een model dat verschillende civilisaties definieert.
* Model verder uitwerken en op papier testen.
* Definiëren van scenario's op basis van het opgestelde model.
* Het op papier zetten van mechanics betreft AI binnen Skies of Saturn.
* Het bouwen van een prototype
* Verschillende entiteiten bedenken en testen.

##### Vooronderzoek

Toen ik de filosofie door kreeg, ben ik er anders naar gaan kijken, hoe kun je dit bepaalde gameplay vertalen naar de speler zonder het gebruik van frontend (UI). In de huidige builds is geen AI aanwezig apart van een paar “beestjes” die tevoorschijn komen zodra je in de buurt komt van een “spawnpoint”. Een spawnpoint is een willekeurige positie in de wereld. Elk blokje waarop je crasht, groeit een boom uit. Het idee hierachter is door dit te doen leven te schenken in jouw digitale wereld. Daarbij was het idee verschillende civilisaties in de wereld te plaatsen. Een civilisatie groeit naarmate jij als speler meer bomen gezamenlijk plaatst. Deze bomen geven een civilisatie energie en groeien vervolgens uit tot een volwaardig zelfstandig ecosysteem.

##### Functioneel ontwerp

Ik ben aan de slag gegaan met het opstellen van verschillende scenario’s die uitgespeeld kunnen worden. Een civilisatie heeft resources nodig om verder te kunnen groeien, dit doen ze door uit hun leefgebied resources te verzamelen. Per cycli komen er resources bij afhankelijk hoe goed ze zijn met het verzamelen en de hoeveelheid dat beschikbaar is.

##### Technisch ontwerp

Voor deze sprint was een technisch ontwerp niet van toepassing.

##### Bouw

Een kolonie is een deel van een civilisatie heeft zijn eigen gedrag in plaats van een enkele entiteit. Daarnaast krijgt een kolonie de mogelijkheid om zich aan te passen zodra er twee koloniën naar elkaar toe groeien. Zodra een kolonie gaat groeien, gaat een kolonie de wereld exploreren met verschillende wezens. Deze wezens hebben hun eigen gedragsregels maar deze zijn van minder belang dan een hele kolonie. Hoe snel een kolonie groeit is afhankelijk van de resources die beschikbaar zijn. De gemiddelde looptijd van een niveau zit rond de drie minuten. Als een kolonie op een bepaald niveau zit kan deze een soort van power-up generen. Deze power-ups worden aan jouw vrijwillig gegeven of deze zou je kunnen stelen met als gevolg dat zo’n kolonie dood kan gaan. Door verschillende soorten koloniën te creëren geeft het spel meer diversiteit. Met als gevolg zodra de speler een nieuw gebied bezoekt en heel ander karakter van de wereld te zien krijgt.

##### Testen

Een scenario testen, deed ik gezamenlijk met Karel. Het testen van zo’n een civilisatie deden we door ze te spelen in een denkbeeldige wereld om te kijken of de uitkomst enigszins zinvol was. Om te beginnen heb ik vier civilisaties gedefinieerd: Military, Tech, Diplomacy en Empire. Deze vier civilisaties hebben ieder unieke karaktereigenschappen. Voor de test heb ik gekozen voor de civilisatie Military. In figuur 2 ziet u een voorbeeld van een aantal karakteristieken. De groene plussen en de rode minnen zijn indicatoren hoe snel een bepaald evenement wordt uitgevoerd. Dit kan vanaf 30 seconden oplopen tot een maximaal van vijf minuten. Daarnaast geven we aan waar deze civilisatie is geplaatst. Wij hebben voor de woestijn gekozen, kleur geel.

Met pen en papier zijn we aan de slag gegaan om een scenario uit te spelen. Iedere civilisatie heeft een “starting point” met een aantal entiteiten op basis van de gegeven waardes in het model. Het begin van dit scenario bestaat uit een populatie van tien daarnaast zijn er geen actieve entiteiten aanwezig. Na 15 seconden hebben we een totaal van drie resources en één verkenner. Na een minuut zijn we beland bij niveau twee met de volgende entiteiten: twee verkenners, twee handelaren en negen resources. Het is een eenvoudig hulpmiddel om een model te testen. Conclusie, het was duidelijk dat het aantal gebeurtenissen na bijvoorbeeld drie minuten zeer klein was waardoor de speler niet de indruk zou krijgen dat er veel veranderingen zou plaatsvinden. Een logische vervolgstap was het verlagen van de tijden per update cyclus. Er was geen feedback aanwezig hoe dit zou moeten uitpakken in het eindresultaat omdat het concept van civilisaties in Skies of Saturn nog niet vast stond.

*Het idee betreft civilisaties is niet verder gekomen dan de tekentafel. De focus werdt op de entiteiten zelf gelegd in plaats van grote civilisatie. Het ecosysteem concept werd als key gameplay gezien maar wegens tijdgebrek en de limitaties dat Hence Games oplegde werd het al snel onmogelijk om dit door te zetten.*

|  |
| --- |
| C:\Users\Dynad\Desktop\design_sos_model.png  Figuur |
| Een model dat verschillende soorten civilisaties definieerd. |

#### Sprint 2 – Opzet framework (1 maart tot 7 maart)

Deze sprint staat in het teken van het bedenken van interacties en gedragingen. Om een goed idee te krijgen van het spel in zijn huidige status, heb ik de oorspronkelijke versie van Skies of Saturn nagebouwd. De oorspronkelijke versie is de versie die Hence Games op dat moment aanleverde.

De wereld zou bestaan uit stukken landschap waartussen je heen kunt vliegen, in plaats van dat je het speelveld vanaf boven ziet, is het juist het tegenovergestelde. Je ziet namelijk de wereld vanaf het gebergte naar boven toe, daarom zie je de wolken als achtergrond langzaam meebewegen. Doordat Skies of Saturn een exploratie spel is, hebben we gekozen om de wereld willekeurig te laten genereren. Oftewel, de wereld ziet er per speelsessie anders uit.

Werkzaamheden voor deze sprint zijn als volgt:

* Begin framework voor prototyping
* Noise map generatie

##### Vooronderzoek

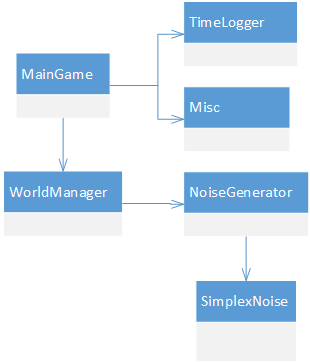
Om willekeurige werelden te kunnen genereren heb je een datasource nodig. Een datasource is in dit geval een noise map[[19]](#footnote-19). Een noise map kun je het beste vergelijken met een afbeelding met willekeurig ruis, zie afbeelding hieronder. Je hebt verschillende soorten noise maps: 1D, 2D en 3D. Voor Skies of Saturn is een 2D map gebruikt. Door gebruik van noise maps is het mogelijk om per speelsessie hetzelfde te genereren door middel van een seed. Een seed is een vaste waarde dat aan het begin van de procedure wordt ingevoerd. Als deze waarde nooit verandert, zul je altijd dezelfde map blijven genereren.

|  |  |
| --- | --- |
| noise.gif  Figuur 3 | Perlin_noise.jpg  Figuur 4 |
| Non-coherent noise in grijswaardes (0, 255) | Perlin/Simplex noise |

##### Functioneel ontwerp

Ik heb het framework compact gehouden omdat er anders meer gedaan zou zijn dan deze sprint zou suggereren. In figuur 5 ziet u een UML klassendiagram met een aantal definieerde klassen en deze zal ik in het kort toelichten:

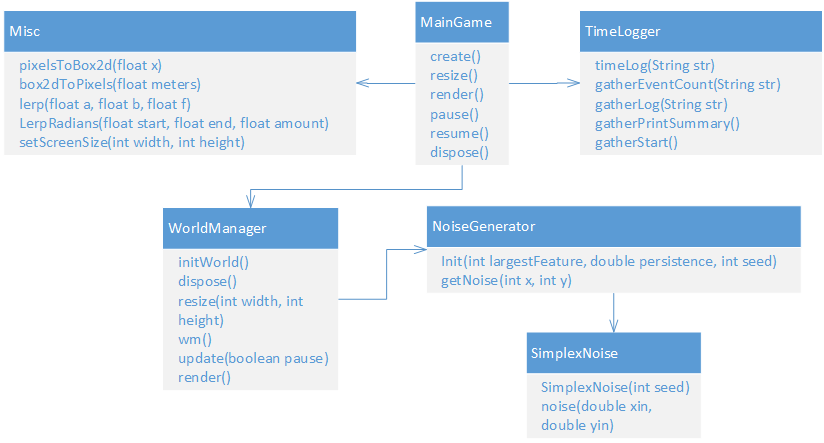
|  |  |
| --- | --- |
| MainGame | Game wordt gestart met deze klasse |
| TimeLogger | Wat de naam al suggereert, houdt de tijd bij tussen frames |
| Misc | Alle overige functies zoals window size |
| WorldManager | Hier gebeurt alle magic, de WorldManager beheert de wereld. |
| NoiseGenerator | Deze klasse is in staat een 2d plaatje te genereren op basis van een stel parameters met als resultaat zie figuur 4 en figuur 7 |
| SimplexNoise | Dit is de klasse die het daadwerkelijke werk doet betreft het genereren van een noise map. |



Figuur 5 – Basis framework met daarbij noise generation toegevoegd.

##### Technisch ontwerp

Het genereren van ruis (noise) heb ik het algoritme Simplex[[20]](#footnote-20) gebruikt. Dit algoritme geeft een lijst terug van waardes tussen de 0.0 en 1.0. Dit alleen, is niet voldoende omdat het non-coherent is (figuur 3). Hiervoor heb ik een klasse NoiseGenerator geschreven die van deze ruis, vormen creëert (zie figuur 4).



Figuur – Extended klassendiagram

##### Bouw

Hieronder ziet u een afbeelding van een willekeurig (procedureel) genereerde map gerenderedt naar pixels met een aangepaste threshold.

|  |
| --- |
| C:\Users\Dynad\Desktop\images sos\noisemap.png  Figuur |
| Procedural generated, Simplex Noise. |

##### Testen

Deze test is visueel getest omdat deze het beste te testen hoe het als resultaat eruit ziet, een unit test zou hier niet goed zijn werking kunnen verrichten door alleen waardes te controleren of deze correct zijn of niet.

|  |  |
| --- | --- |
| Naam | Noise maps |

|  |  |
| --- | --- |
| Getest door: | Danny van den Bree |
| Getest op: | PC, Windows, Java |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Test case beschrijving | Verwachte resultaat | Daadwerkelijke resultaat | Opmerkingen |
| 1 | Noise map gegenereerd | Coherent noise map | PASS | geen |

#### Sprint 3 - Physics en world generation (7 maart tot 14 maart)

Voordat er gewerkt kon worden aan kunstmatige intelligentie is een wereld met verschillende entiteiten noodzakelijk waarmee interacties verricht kunnen worden. Eerste stap was het vertalen van de noise map dat vooraf gegenereerd werd naar de tiles die vervolgens via de renderer op het scherm worden getoond. De noise maps worden niet lokaal opgeslagen maar direct in de renderer geïnjecteerd zodat een oneindige wereld realiseerbaar werd. Dit betekent dat het niet noodzakelijk was om met de hand een wereld te designen. Het enige wat nog op het programma stond waren de individuele textures die op basis van de noise map grijstintenniveaus in de wereld geplaatst konden worden.

Voor physics heb ik de library Box2D gebruikt omdat deze met de engine LibGDX werd meegeleverd. Per tile wordt er een physical box aangemaakt, dit zou qua performance niet heel erg efficiënt zijn omdat je anders oneindig veel Box2D objecten moet aanmaken. Daarom maken we gebruik van pooling. Pooling is het aanmaken van objecten die recyclet kunnen worden.

##### Vooronderzoek

De wereld zou nu vertaald moeten worden naar visuele assets op basis van de noise map, nadeel van deze methode is dat de wereld een vierkantachtige look krijgt. De wereld is een vierkant raster, de indeling van de wereld bestaat uit ofwel rotsen of lucht. Dit wordt per pixel van de noise map gegenereerd. Elke pixel heeft zijn eigen grijswaarde (RGB), 255 is lucht en 0 is een rots. Daarnaast kunnen de grijstinten ook het soort rots bepalen.

De wereld wordt procedureel gegeneerd waardoor het niet mogelijk is om oneindig veel physical bodies aan te maken. Doordat alle tiles dezelfde grootte krijgen, heb ik er voor gekozen om hier een pooling pattern toe te passen. Pooling is het aanmaken van een maximaal aantal objecten die verdeeld worden over het aantal aanvragen per update loop. Teveel aanvragen kan resulteren dat er tiles aanwezig zijn die geen body hebben en kan hierdoor geen collision plaatsvinden. Je zou als speler in worst case scenario door landschap heen kunnen vliegen. Door dit te testen bleek 200 bodies voldoende te zijn.

##### Functioneel ontwerp

Eén tile in de wereld zal gelijk zijn aan één pixel van de noise map. Voor alle mogelijke combinaties zijn er een totaal van 16 verschillende tiles aanwezig.

|  |
| --- |
| http://www.saltgames.com/wp-content/uploads/2009/12/tileset.png  Figuur 8 |
| Er zijn 16 tiles in de tileset die ieder een eigen ID nummer hebben. |

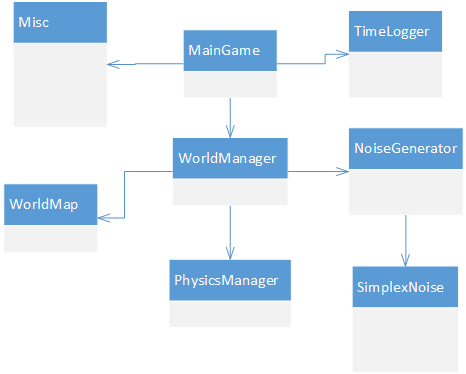
Om te kunnen bepalen welke texture per tile hoort, kijken we naar de aangrenzende tiles. In plaats van dat er lange complexe condities worden geschreven, geven we een score per richting van een tile (zie figuur 10). Als er een aangrenzende tile aanwezig is, tellen we dat op bij het totaal. Als er bijvoorbeeld alleen een tile erboven aanwezig is, geven we het een score van “1”. Als er boven en onder een tile aanwezig is, dat is wordt de score 1 + 4 = **5**.

|  |  |  |  |
| --- | --- | --- | --- |
| http://www.saltgames.com/wp-content/uploads/2009/12/filledGrid.png  Figuur 9 | http://www.saltgames.com/wp-content/uploads/2009/12/edge-values.png  Figuur 10 | http://www.saltgames.com/wp-content/uploads/2009/12/numberedGrid.png  Figuur 11 | http://www.saltgames.com/wp-content/uploads/2009/12/floatingIsland.png  Figuur 12 |
| **Stap 1** | **Stap 2** | **Stap 3** | **Stap 4** |
| De map, procedureel gegenereerd m.b.v. de NoiseGenerator | Score van een enkele tile | De getallen zijn de scores berekend op basis van stap 2 | Resultaat, vertaling naar een tileset |

Nadat alle tiles gecontroleerd zijn die binnen het bereik van de renderer vallen, creëren we het volgende beeld, **stap 3**. Zodra een waarde aan elke tile is toegekend, zoeken we de score op in de tileset met als resultaat zie **stap 4**.

Voor deze sprint zijn er een paar klassen bijgekomen die noodzakelijk zijn voor het gebruik van de tileset en Box2D. In figuur 13 ziet u een UML klassendiagram met een aantal nieuwe definieerde klassen en deze zal ik in het kort toelichten:

|  |  |
| --- | --- |
| WorldMap | In deze klasse worden de pixels van de noise map vertaald naar tiles (textures). |
| PhysicsManager | Deze klasse maakt de physical bodies aan voor de tiles. |



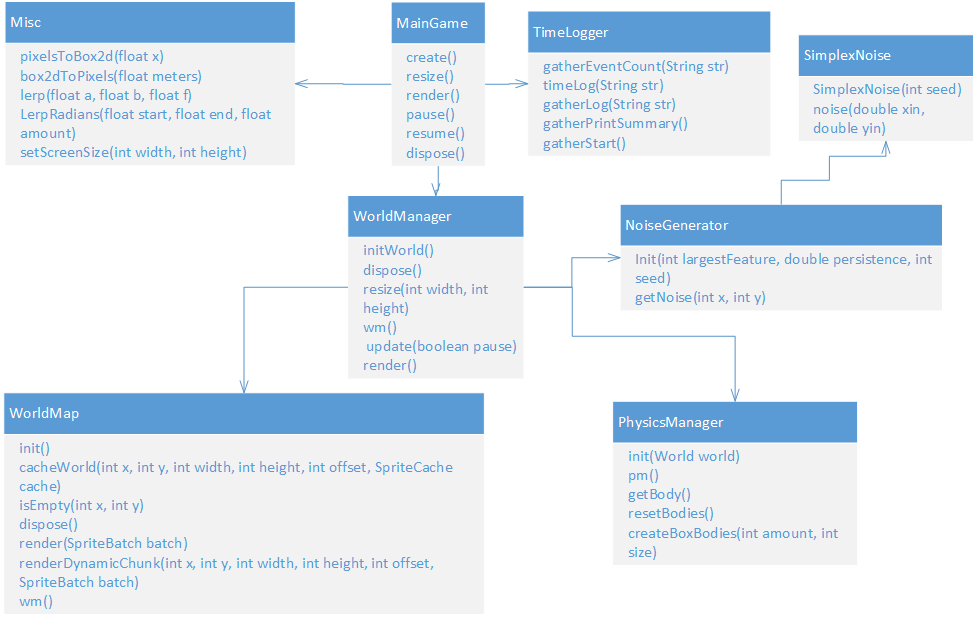
Figuur

##### Technisch ontwerp

Per pixel wordt er aangeven aan welke waarde het aan moet voldoen. Als een pixel voldoet aan bijvoorbeeld minimale waarde van 0.2 dan wordt deze vertaald naar een tile (blokje). Vervolgens is het de bedoeling dat de wereld, nu een vierkant raster wordt vertaald naar een tileset om het meer op een landschap te laten lijken. Ik heb hiervoor een algoritme ontwikkeld dat tijdens het renderen van de tiles afgehandeld wordt zodat er geen overhead ontstaat. Je wilt zo min mogelijk geheugen gebruiken omdat mobiele apparaten daar een beperking in hebben.

Om physical bodies aan te kunnen maken heb ik de PhysicsManager klasse geïntroduceerd. De functie createBoxBodies maakt een lijst met aangemaakte bodies met een vooraf gegeven grootte. Daarnaast heb ik een getter functie het leven ingeblazen, zodat je op elk moment een physical body kan vergaren, als ze er geen bodies beschikbaar zijn geeft hij een NULL terug. Er moet dan gewacht worden op één update cyclus.

Resultaat is dat er uiteindelijk een totaal van 200 bodies beschikbaar zijn voordat het spel begint. Omdat landschap in het algemeen niet verplaatst, zijn de bodies gedefinieerd als static. Waardoor ze vlot verwerkt kunnen worden omdat voor statics geen velocity, snelheid en beweging berekend hoeft te worden.



Figuur

##### Bouw

Op figuur 15 ziet u twee afbeeldingen waarvan links de standaard manier wordt gehanteerd met één type tile. Waarin het andere, gebruik maakt van 16 verschillende tiles waardoor je oneven randen krijgt.

|  |
| --- |
| C:\Users\Dynad\Desktop\skiesofsaturn_bitwise.png  Figuur |

##### Testen

In mijn prototype was ongeveer 200 vooraf aangemaakte boxen voldoende om de wereld die binnen het frustum van de camera in te vullen. Het testen werdt visueel gedaan omdat het verwachte resultaat eenvoudig waarneembaar was.

|  |  |
| --- | --- |
| Naam | *Tileset* |

|  |  |
| --- | --- |
| Getest door: | Danny van den Bree |
| Getest op: | PC, Windows, Java |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Test case beschrijving | Verwachte resultaat | Daadwerkelijke resultaat | Opmerkingen |
| 1 | Noise map vertalen naar een tileset | Dat alle scores vertaald worden naar de juiste tile | Passed | Geen |
| 2 | Physical bodies, 200 in totaal | Dat iedere tile een physical body krijgt | Passed | Geen |

#### Sprint 4 - Player mechanics (14 maart tot 1 april)

Zonder de speler is het testen van een bepaalde mechaniek een lastig verhaal. Daarom heb ik besloten om het hoofdkarakter na te maken. Niet alleen het visuele aspect is van groot belang, ook komt het gameplay gedeelte aan bod waar veel tijd aan werdt besteedt. We wilden een zo goed mogelijke besturing ontwikkelen dat het voor de speler op een natuurlijke manier aanvoelt. Met de huidige besturing, dat op constante snelheid beweegt ongeacht welke directie je ook opvliegt voelt oninteressant aan. Daarom heb ik besloten hetzelfde soort besturing die Hence Games heeft ontwikkeld over te nemen en daarop verbeteringen uit te voeren.

Voor deze sprint heb ik acceleratie met verschillende snelheden van de speler toegevoegd. Deze zal basis van de windrichting beinvloed worden waardoor het schip aantrekkelijk aanvoelt om te besturen. Als speler vlieg je met je schip op de wind. Om wind te kunnen simuleren worden er willekeurig particles aangemaakt die naar een bepaalde hoek voortbewegen. De speler krijgt de mogelijkheid door de wind meer snelheid op te pakken. Je kunt ook de wind gebruiken om af te remmen. De vleugels van het schip geeft de speler een indicatie hoeveel wind op dat moment wordt opgevangen. De wind particles hebben een korte levensduur waardoor het niet te eentonig wordt. Met een vooraf gedefinieerde timer wordt de windrichting aangepast.

Werkzaamheden voor deze sprint zijn:

* Besturing speler
* Wind particles
* Windrichting, de speler krijgt snelheid op basis van de windrichting
* WorldCamera

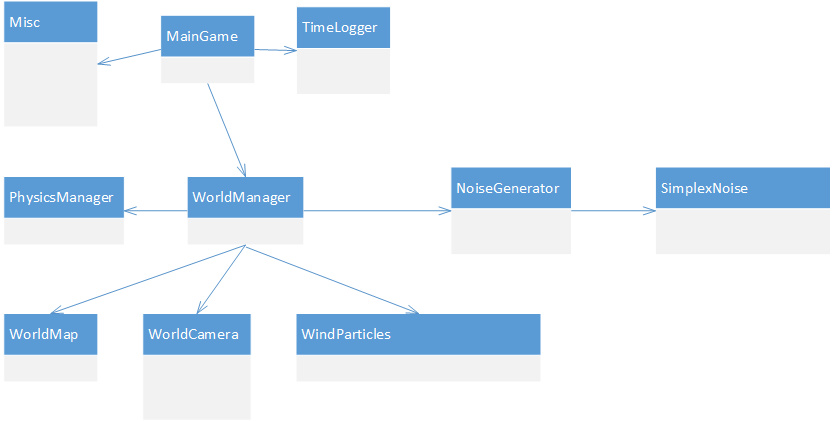
##### vooronderzoek

Geen vooronderzoek noodzakelijk.

##### Functioneel ontwerp

Voor deze sprint zijn er een paar klassen bijgekomen die noodzakelijk zijn voor het gebruik van de tileset en Box2D. In figuur 13 ziet u een UML klassendiagram met een aantal nieuwe definieerde klassen en deze zal ik in het kort toelichten:

|  |  |
| --- | --- |
| WorldCamera | Volgt de speler in een vloeiende beweging. |
| WindParticles | Particles die wind simuleren. |



De acceleratie van het schip wordt geregeld d.m.v. de hoek waarop het schip vaart. Als je in dezelfde richting opgaat als de wind zul je sneller vaart oppakken. Je kunt afremmen door tegen de wind in te vliegen. Er is geen zwaartekracht aanwezig omdat de pov (player of view) van bovenaf is. Hierdoor als je met je schip bijna stil staat alsnog de kant waar de wind op staat wordt opgedreven. Dit is de zwaartekracht in werking, omdat er geen boven of onder is heb ik dat verplaatst naar windrichting. Je stuurt het schip door je muis te verplaatsen naar de hoek van het scherm waar de speler naar toe wilt vliegen (pc versie). Daarnaast door linker muisknop te gebruiken klap je de vleugels in en kun je met dezelfde snelheid door de wereld blijven manoeuvreren. De snelheid neemt minder snel af zodra de speler tegen de wind in zou opvliegen.

In de onderstaande tabel zijn de verschillende animaties weergegeven met daarbij de textures die gebruikt gaan worden. Een tileset is opgebouwd uit opgesplitste textures.

|  |  |  |
| --- | --- | --- |
| E:\Dropbox\SoS intern\gfx\wing2.png  Figuur 16 | J:\Dropbox\SoS intern\gfx\wing2_army.png  Figuur 17 | J:\Dropbox\SoS intern\gfx\wing2_black.png  Figuur 18 |
| **Texture tileset (Marine)** | **Army** | **Dark** |
| E:\Dropbox\Screenshots\ship_wing_open_max_wind.png  Figuur 19 | E:\Dropbox\Screenshots\ship_wing_open_less_wind.png  Figuur 20 | E:\Dropbox\Screenshots\ship_wing_closed.png  Figuur 21 |
| **Wing open, wind mee.** | **Wing open, wind tegen.** | **Wing gesloted, ongeacht stand van de wind.** |

Niet altijd zijn er grote veranderingen noodzakelijk, ook kleine aanpassingen hebben een impact. Een voorbeeld hiervan is dat de camera in en uitzoomt op basis van jouw snelheid, des te sneller je gaat des te verder de camera uitzoomt.

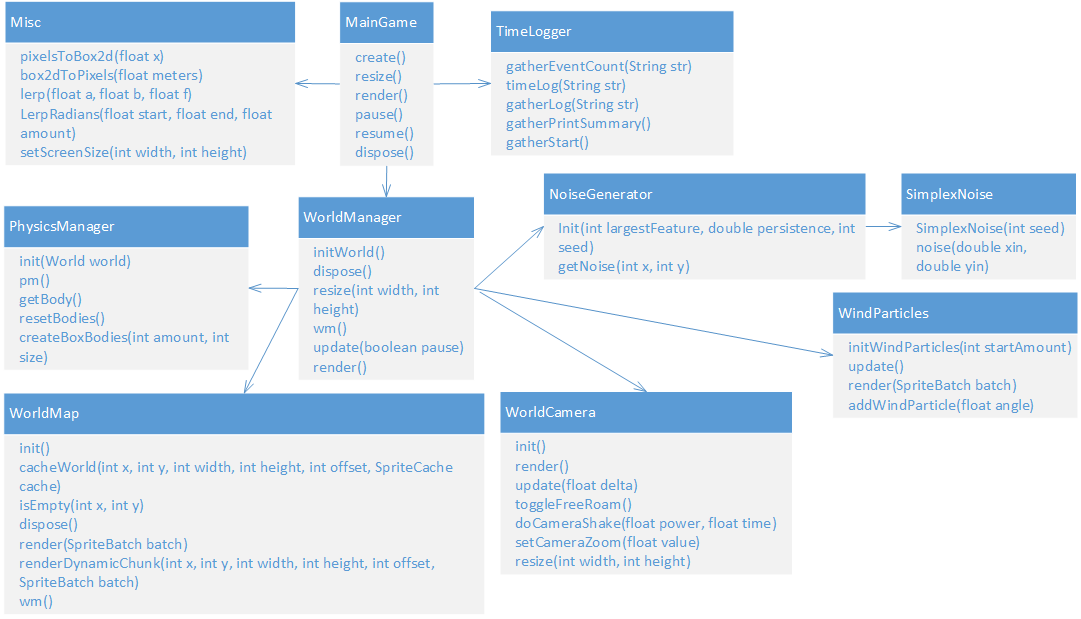
Windparticles simuleren de windrichting d.m.v. een bepaalde richting op te bewegen. Ze zullen gezamenlijk één kant opbewegen zodat de speler dat als indicator kan gebruiken in welke richting de meeste snelheid behaald kan worden. Daarnaast wordt de richting periodiek aangepast.

Ik heb eerst een aantal variabelen bedacht waaronder minimale- en maximale snelheid, acceleratie op basis van welke wing actief is. De waardes in het onderstaande tabel worden in units gedefinieerd omdat deze waardes relatief zijn.

|  |  |
| --- | --- |
| Eigenschap | Waarde |
| Startsnelheid | 6 units |
| Minimale snelheid | 3 units |
| Maximale snelheid | 15 units |
| Minimale hoek voor het oppakken van wind | 216 graden |
| Snelheid overgang tussen langzaam en snel | 10 units |
| Snelheid deceleratie wing closed (op lage snelheid) | 0.01 units |
| Snelheid deceleratie wing closed (op hoge snelheid) | 0.025 units |
| Snelheid deceleratie wing open (op lage snelheid) | 0.05 units |
| Snelheid deceleratie wing open (op hoge snelheid) | 0.1 units |
| Snelheid acceleratie wing closed | 0.01 units |
| Snelheid acceleratie wing open | 0.1 units |

##### Technisch ontwerp

De uitvoering hiervan heb ik twee klassen geschreven Windparticle en WindParticles. De WindParticle is houdt informatie vast over zijn positie, rotatie, kleur, grootte en snelheid. De WindParticles klasse maakt deze WindParticle objecten aan met een willekeurige positie en grootte, snelheid van de rotatie etc. Ik heb voor een limiet van een maximaal van ongeveer 75 WindParticle objecten, dit zorgt voor voldoende dichtheid. Met als resultaat dat er willekeurig op het scherm windparticles geplaatst worden. Die vervolgens een richting opgaan waarop de vleugels van de speler reageren. Daarnaast wordt ook de snelheid van speler aangepast op basis hoe dicht de hoek op elkaar zijn afgestemd.

****

Figuur De klassen WindParticles en WorldCamera toegevoegd.

##### Bouw

De animaties voor de vleugel heb ik in Photoshop getekend, deze zijn opgesplitst en vervolgens in Spine geïmporteerd. De reden dat ik ze heb opgesplitst is, omdat Spine werkt op basis van skeleton animaties. Dit houdt in dat iedere texture aan een bone vastzit. Nadat de twee animaties wing open en closed af waren zijn ze geëxporteerd naar een Json formaat dat vervolgens ingelezen kon worden door mijn eigen geschreven functie. Op basis van de hoek tussen de speler en de windrichting wordt het juiste frame van de animatie weergegeven. Doordat ik niet met dezelfde engine werkte als dat van Hence Games had ik geen speler animaties dat ik kon gebruiken van Hence Games omdat de systemen van elkaar afweken. Om deze reden heb ik de animaties zelf nagemaakt in het programma Spine.

|  |
| --- |
| E:\Dropbox\Screenshots\Screenshot 2014-06-04 23.59.13.png  Figuur 23 |
| In Spine heb ik de animaties gemaakt, open en closed. |

|  |
| --- |
| E:\Dropbox\Screenshots\windparticles.png  Figuur |
| De blauwe ruiten bewegen een richting op, simuleren de windrichting. |

##### Testen

Voor deze sprint geld hetzelfde, de kwaliteit werdt getest d.m.v het visueel te testen door één van mijn collega’s. Dit werdt vervolgens goedgekeurd door de projectmanager en kon de volgende sprint van start gaan.

|  |  |
| --- | --- |
| Naam | *Wind particles* |

|  |  |
| --- | --- |
| Getest door: | Danny van den Bree |
| Getest op: | PC, Windows, Java |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Test case beschrijving | Verwachte resultaat | Daadwerkelijke resultaat | Opmerkingen |
| 1 | WindParticles windrichting correct | De wind particles gezamenlijk naar een bepaalde richting bewegen en dat de vleugels van de speler daarop anticipeerd. | Passed | geen |

#### Sprint 5 - front end en Tree generation (1 april tot 7 april)

Tijdens het bouwen van verschillende prototypes is aan mij gevraagd een debug UI in te bouwen om bepaalde settings in het spel aan te kunnen passen. Een voorbeeld hiervan is het aan kunnen passen van de minimale en maximale snelheid, acceleratie of deceleratie.

Doordat alle content in de wereld willekeurig geplaatst worden verlies je flexibiliteit. Hierdoor kun je alleen variabelen introduceren die bijvoorbeeld de noise map een andere look geven. Je kunt namelijk de afmeting veranderen waardoor je meer detail krijgt met als nadeel dat het geheel kleiner wordt. Te klein heeft als gevolg dat één blokje gelijk staat aan een groep pixels van de noise map waardoor het detail van de noise map verloren gaat.

Het idee van Skies of Saturn is dat je met je schip rondvliegt in een onbekende wereld waarin geen leven aanwezig is. Jij krijgt de rol als speler om bomen te gaan planten door te crashen op willekeurige rotsblokken. Hieruit groeit een Tree die over tijd groter zal worden en zal energie opwekken waar civilisaties vervolgens gebruik van maken. Een manier dit te visualiseren door deze bomen met enkele eenvoudige animaties te plaatsen waar je als speler bent gecrasht.

Een Tree kan alleen geplaatst worden zodra je voldoende resources binnen hebt gehaald door langs stukken landschap te vliegen. Er zit een tussentijdse timer die bijhoudt wanneer je de mogelijkheid krijgt opnieuw een Tree te spawnen, deze wordt bijvoorbeeld gereset na tien seconden. Als je voor die tijd op een willekeurige plaats crasht wordt je bij de laatste geplaatste Tree teruggezet.

Doel is om een volgroeide Tree te visualiseren door gebruik te maken van een recursief fractal in combinatie met een boomstructuur bijvoorbeeld B-Trees, H-Trees of Pythagoras-Trees. Oftewel een L-system, dit systeem wordt gebruikt om fractalachtige vormen te beschrijven. Door te verhogen van het recursief niveau wordt de complexiteit van de vorm groter. Dit creëert de illusie dat een tree een groei proces ondergaat. Het algoritme heb ik zelf ontwikkeld op basis van bestaande technieken.

Werkzaamheden voor deze sprint zijn:

* Tree generation
* Speler -> death -> recycle -> spawn Tree chain
* UI Test platform

*De functionaliteit omtrent tree generation is overigens uit het spel geschrapt, de speler crashed niet meer om zelf bomen te planten maar in plaats daarvan neemt het ecosysteem deze mechanic over. De wezens die rondvliegen gaan op zoek naar glyphs en brengen deze vervolgens terug naar huis om een Tree verder te laten groeien. Dit heb ik overigens niet verder uitgewerkt wegens tijdgebrek.*

##### vooronderzoek

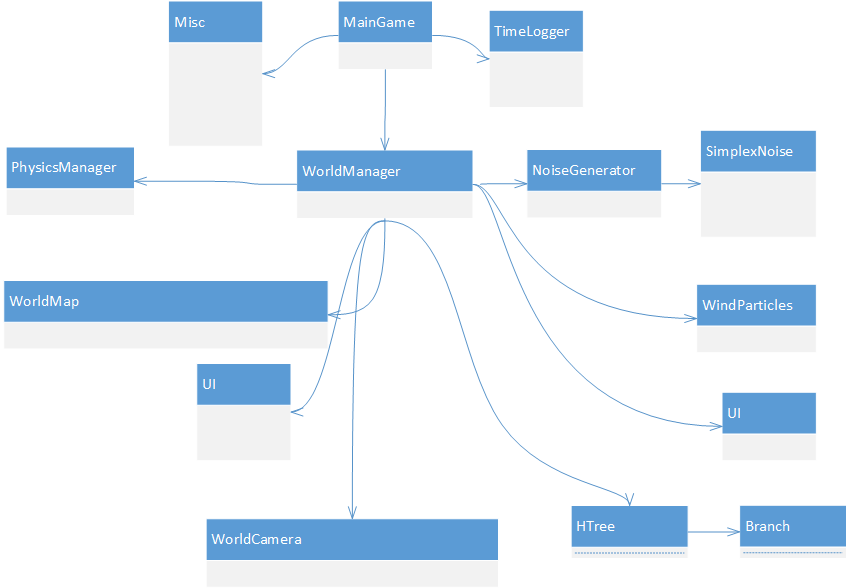
Voor de bomen die geplant kunnen worden door de speler heb ik gekozen voor een H-Tree omdat deze efficiënt en vlot te genereren is. Omdat een H-Tree alleenstaand niet overeen komt met een echte boom heb ik een andere variant ontwikkeld die op basis van een H-Tree de visuele kant heeft van een B-Tree. Een H-Tree splitst elke node gelijkwaardig, een hoek van 90 graden. Door een extra variabel dat een willekeurig getal vasthoudt te vermenigvuldigen met de 90 graden, krijg je per branch een andere hoek. Met als resultaat krijg je het volgende, zie afb. De trees zijn opgebouwd uit eenvoudige lijnen met een bepaalde dikte. Nadeel hiervan dat dit zwaar valt voor mobile devices, na ongeveer het plaatsen van 3 of meer trees de performance in een rap tempo achteruit gaat. Dit komt omdat elke branch apart naar de GPU gestuurd moet worden, zodra je een recursief van 5 of meer had ingesteld krijg je een totaal van maarliefst 63 branches ( 1 + 2 + 4 + 8 + 16 + 32).

|  |  |  |
| --- | --- | --- |
| http://www.cs.princeton.edu/courses/archive/fall05/cos126/assignments/htree3.png  Figuur 25 | http://blog.press.princeton.edu/wp-content/uploads/2011/07/trees.jpg  Figuur 26 | File:Pythagoras tree 1 1 13 Summer.svg  Figuur 27 |
| H-tree | B-tree | Pythagoras-tree |

##### functioneel ontwerp

Het verbeteren van de performance heb ik een andere methode gebruikt, namelijk een spritebatch. In plaats van aparte lijnen, sprites te gebruiken, sprites zijn niets anders dan een in geheugen geladen afbeeldingen. Omdat er een enkele sprite gebruikt wordt kun je deze in een batch renderen. Een batch stuurt een sprite eenmalig naar de GPU met daarbij horende size en coördinaten. Dit zorgt voor een enorme snelheidswinst met als resultaat in plaats van 3 bomen tot wel 20 bomen geplaatst kunnen worden zonder enige performance verlies.

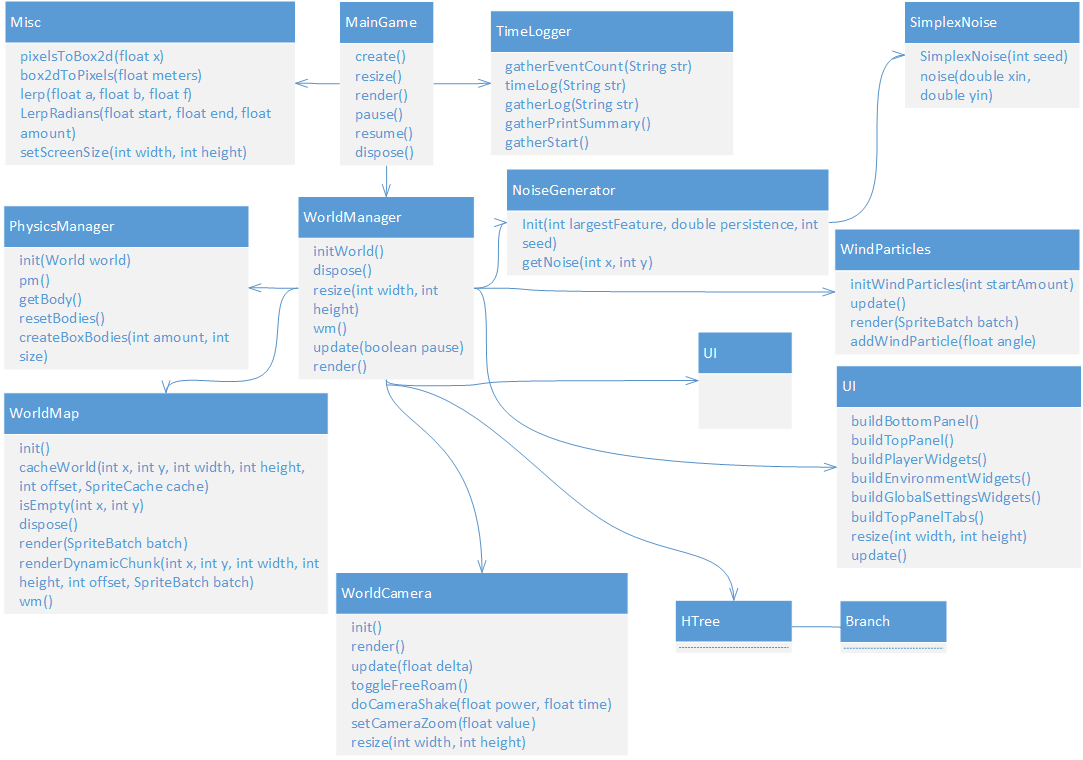
Omdat er wind aanwezig is, komen statische bomen niet tot zijn recht. Door het toevoegen van een kleine beweging geeft het een realistischer gevoel. Het aanpassen van de positie en rotatie per branch (sprite) is een zware operatie. Dit heb ik opgelost door per branch de parent node te onthouden. Als er een parent node beschikbaar is, vraag de centerpositie op. Omdat de child node in het midden van de parent node geplaatst wordt, blijven alle nodes in harmonie samen bewegen. De positie van de branches worden niet direct aangepast, in plaats daarvan passen we de rotatie aan van een branch.



##### technisch ontwerp

De algoritme gaat als volgt, per branch wordt de onderkant geroteerd want een rotatie om zijn centrale as geeft een ongewenst resultaat. Om een oscillatie effect te creëren gebruiken we de cosinus met een delta waarde. Dat vermenigvuldigen we met de waarde PI omdat we in een cirkel een rotatie uitvoeren. Vervolgens vermenigvuldigen we dat met de depth waarde. De depth waarde ofwel “level” is hoe diep de branch node in de gehele tree zich bevind. De root heeft de waarde van 1 en een leaf node als er een recursief van 5 gekozen is, een level waarde van 6. De reden van het level systeem is, omdat de bladeren in het algemeen meer bewegen dan de stam van de boom.

|  |
| --- |
| (Cos(delta) \* PI) \* (current branch level / SPEED) |



###### bouw

Voor deze sprint is er een UI gerealiseerd waarmee je direct waardes van de game kon aanpassen. Voor game designers is dit zeer handig, je hoeft namelijk niet voor elke aanpassing terug naar de code.

|  |
| --- |
| E:\Dropbox\Screenshots\ui_settings_player_01.png  Figuur |
| **Speler settings** |

|  |
| --- |
| E:\Dropbox\Screenshots\ui_settings_general_01.png  Figuur |
| General settings |

|  |
| --- |
| E:\Dropbox\Screenshots\sos_trees.png  Figuur |
| **Fractal H-Tree *De bladeren worden op de kruising van de branches geplaatst.*** |

##### testen

Ook voor deze sprint geld dat alles visual op kwaliteit werdt getest, de bomen zijn aanvankelijk erbij gekomen omdat Hence Games problemen had met het generiek kunnen genereren van bomen op de mobiele platformen. De UI hoeft niet getest te worden omdat deze geen deel uitmaakt van het spel en alleen voor debug doeleinde is gebruikt.

|  |  |
| --- | --- |
| Naam | *Trees* |

|  |  |
| --- | --- |
| Getest door: | Danny van den Bree |
| Getest op: | PC, Windows, Java |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Test case beschrijving | Verwachte resultaat | Daadwerkelijke resultaat | Opmerkingen |
| 1 | Spawnen van een Tree op locatie | Een tree spawnen op locatie en vervolgens geanimeerd op de wind meebewegen | Passed | Geen |
| 2 | Tree animatie | Het bewegen van bomen d.m.v. wind | Passed | Geen |

#### Sprint 6 - Post processing & Parallax background (7 april tot 14 april)

Nadeel van willekeurig gegenereerde wereld is het ontbreken van variatie in het kleurencontrast. Je hebt er een aantal technieken voor om dit te verhelpen, en één daarvan is “postprocessing”. Postprocessing is in feite een filter dat je over je eind render heen legt. Dit geeft een “after effect” zoals grayscale, edge detection of een pixelizer. Voor Skies of Saturn heb ik een drietal shader types gebruikt: vignette, grayscale en sepia. Vignette geeft effect waardoor de randen van het scherm iets donkerder zijn. Het grayscale gedeelte zorgt ervoor dat alle kleursoorten bijvoorbeeld de tiles een grijze tint krijgen. Als laatst, de sepia shader, deze shader kun je een bepaalde kleurtint meegeven waardoor alle tiles gezamenlijk één geheel wordt.

Werkzaamheden voor deze sprint zijn:

* Vignette shader
* Sepia shader
* Dynamische shader achtergrond en een Parallax achtergrond

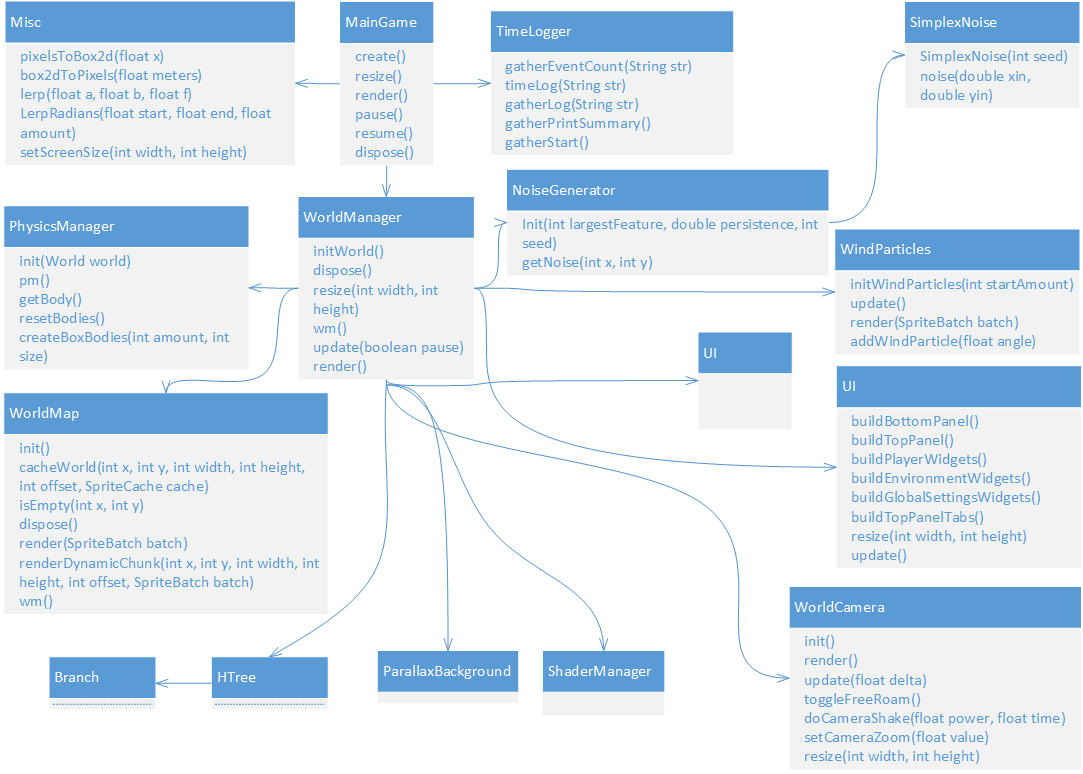
##### Vooronderzoek

Voor de basis ben ik begonnen met het genereren van een achtergrond, puur voor oriëntatie. Achtergrond bestaat uit een aantal figuren, dat in de tegenovergestelde richting van de camera bewegen. Omdat het spel in 2d is, kun je het diepte geven door middel van een parallax achtergrond. Daarnaast heb ik gekeken of shaders een leuke toepassing zou kunnen hebben. Shaders zijn stukken code dat geavanceerde visualisatie mogelijk maakt. Dit stuk programmeercode wordt in de taal GLSL geschreven omdat we gebruik maken van de bibliotheek OpenGL. De tegenhanger van OpenGL is DirectX en heeft HLSL als standaard taal voor zijn shaders. Er zijn twee soorten shaders: Vertex- en Fragmentshaders. De Vertex shader vertegenwoordigd de manier waar en hoe de afbeeldingen getoond worden. Deze shader is bedoeld om de vertices van een shape te manipuleren. Elke afbeelding die je naar de GPU stuurt heeft een totaal van 4 vertices. Nadat de vertex shader zijn werk heeft gedaan kom je terecht bij de fragment shader. De fragment shader kan de pixels van een afbeelding manipuleren, bijvoorbeeld alle pixels van de afbeelding moeten nu allemaal de kleur rood zijn.

##### Functioneel ontwerp

Het spel speelt zich af in de lucht van Saturnus. Waarvan het beeld vanuit de oppervlakte naar de horizon staat. Voor de parallax achtergrond heb ik gekozen voor 3 lagen in verschillende maten. Van deze 3 lagen definieer ik het type afbeelding, de hoeveelheid afbeeldingen gelijktijdig aanwezig mogen zijn en hun levensduur. Deze lagen zijn actief binnen een bereik van anderhalf keer het scherm resolutie. Omdat het anders vreemd overkomt zodra deze achtergronden altijd middenin het scherm opkomen en verdwijnen.

##### Technisch ontwerp



##### Bouw

|  |
| --- |
| E:\Dropbox\Screenshots\parallax_background.png  Figuur 31 |
| Verschillende background afbeeldingen die sequentieel elkaar overlappen. Ze bewegen in de tegenovergestelde richting van de speler, dit creëert een leuk diepte effect. |
| E:\Dropbox\Screenshots\vignette_sepia.png  Figuur 32 |
| Vignette en Sepia shader |
| C:\Users\Dynad\Desktop\background.png  Figuur 33 |
| Dynamische geanimeerde achtergrond shader |

##### testen

|  |  |
| --- | --- |
| Naam | *Shaders* |

|  |  |
| --- | --- |
| Getest door: | Danny van den Bree |
| Getest op: | PC, Windows, Java |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Test case beschrijving | Verwachte resultaat | Daadwerkelijke resultaat | Opmerkingen |
| 1 | Parallax background effect | Verschillende type achterground die overelkaar heen bewegen op basis van de camera positie. | PASS | geen |

#### Sprint 8 - Defining AI types & Parallax background (14 april tot 1 mei)

Om de wereld levendiger te maken, plaatsen we willekeurig entiteiten in de wereld. Entiteiten kunnen zowel vijanden, planten, koloniën of sensoren zijn dat bepaalde delen van de wereld kan beïnvloeden. Deze entiteiten bevatten bepaalde acties (behaviours). Deze acties worden door entiteiten zelfstandig uitgevoerd en dit kunnen we verstaan onder Artificial Intelligence (AI). Door entiteiten bepaalde acties uit te laten voeren zoals het verplaatsen naar een positie; of een vijand wacht in alle stilte totdat een prooi op zijn pad is terechtgekomen en actie onderneemt, geeft de speler de indruk dat de wereld van Skies of Saturn leeft.

##### Vooronderzoek

Er zijn een tal van systemen die gebruikt worden voor het implementeren van kunstmatige intelligentie. Ik heb drie verschillende onderzocht: FSM, Decision Tree en Behavior Tree. FSM staat voor Finite State Machine. Finite State Machine zoals de naam al zegt werkt het op basis van states. Dit houdt in dat er één status per keer geactiveerd mag zijn en dat deze niet gewijzigd mag worden totdat de huidige status is afgerond. Decision Tree, evalueerd nodes totdat één leaf node is bereikt dat aan zijn conditie voldoet en gaat vervolgens terug naar de root. Behavior trees worden op een andere manier geëvalueerd dan Decision trees. Een BT begint bij de root en gaat vervolgens de child selectors evalueren.

###### FSM

Een eenvoudig voorbeeld van een FSM tussen de speler en een willekeurige vijand met de volgende events: Idle, Suspicious, Attack en Flee. Idle kan overgaan naar Suspicious als er een perceptie van de spelers aanwezig is zonder daadwerkelijk de speler te zien. Idle kan direct overgaan op Attack als de speler in zicht is. Suspicious (vijand is actief op zoek naar speler) kan overgaan op Attack als de speler in zicht is. Suspicious kan terug overgaan op Idle na een bepaalde tijd zoeken. Attack kan overgaan op Idle als de speler is gedood. Attack kan overgaan op Flee als zijn leven op het spel staat. Flee probeert te ontsnappen aan de speler, terwijl op hetzelfde moment op zoek is middelen om zijn leven te herstellen. Flee kan overgaan op Suspicious als zijn leven niet meer in gevaar is.

|  |
| --- |
| Figuur |
| Een finite state system cyclus[[21]](#footnote-21) |

###### DT

Decision Trees[[22]](#footnote-22) worden geëvalueerd van root tot leaf. Voor een decision tree goed te laten werken, moeten de child nodes van elk parent node alle mogelijke beslissingen vertegenwoordigen. Als een node kan worden beantwoord met "Ja, Nee of misschien', moeten er drie child nodes aanwezig zijn. Dit betekent dat er altijd wel een lagere node beschikbaar is totdat een leaf node is bereikt.

|  |
| --- |
| http://i.stack.imgur.com/UqXMK.gif  Figuur |
| Decision Tree, evalueerd nodes totdat één leaf node is bereikt dat aan de conditie voldoet en gaat vervolgens terug naar de root |

###### BT

Behavior trees worden op een andere manier geëvalueerd dan Decision trees. Een BT begint bij de root en gaat vervolgens de child selectors evalueren. De child nodes worden van links naar rechts uitgevoerd. Een selector is een conditie waarop gecontroleerd wordt of deze true of false is. De child nodes zijn gerangschikt op basis van hun prioriteit. Als bijvoorbeeld een leaf node met een event gekoppeld een true geeft wordt deze vervolgens gestart en de tree gereset. Wanneer een node is gestart wordt deze ingesteld op “running”. De volgende keer de tree wordt geëvalueerd gaat hij de hoogste prioriteit nodes eerst af totdat hij een “running” tegenkomt.

|  |
| --- |
| Figuur |
| Behavior Tree, evalueerd alle leaf nodes totdat één aan de juiste conditie voldoet en start de routine. |

##### Functioneel ontwerp

Er is een eenvoudige versie van een entiteit ontwikkeld op basis van de decision tree die overigens wel is overgezet naar de build van Hence Games. Dit entiteit vliegt rond en zoekt naar glyphs en als jij als speler in de buurt komt van dit entiteit dan probeert hij jouw glyphs af te pakken om terug mee te nemen naar zijn basis. Bij zijn basis dropt hij de glyph en groeit vervolgens een abstract boompje uit. Dit entiteit is overigens getoond op het A-Maze gaming event in Berlijn en heeft tot leuke reacties geleid.

##### Technisch ontwerp

Het is jammer genoeg bij onderzoek gebleven wegens tijdgebrek.

##### Bouw

|  |
| --- |
| C:\Users\Dynad\Desktop\stealer.png  Figuur |
| Stealer entiteit decision tree voor a-maze[[23]](#footnote-23) build |

##### testen

Het ecosysteem moest getest zijn voordat deze uiteindelijk in de Unity build van Hence Games verwerkt kon worden. Bij het testen heb ik vooral gekeken naar de keuzes die de wezens maken op basis van de gegeven informatie die zij tot hun beschikking hebben. Door deze keuzes te toetsen wordt er puur gekeken of deze keuzes die gemaakt worden wel gefundeerd zijn. Keuzes die gemaakt worden kunnen weleens fout worden beschouwd volgens de speler maar dat hoeft niet zo te zijn want als de gemaakte keuze uiteindelijk de beste optie bleek te zijn kun je het alsnog zien als een succes. Overige gameplay zoals de besturing van de speler heb ik door een collega laten testen omdat deze niet noodzakelijk door een unit test gehaald hoeft te worden.

Het is van belang dat de functionaliteit van het ecosysteem wordt getest. Hieronder vallen alle mogelijke handelingen die een entiteit kan uitvoeren. Het testen van de functionaliteiten binnen het ecosysteem moest er vooral gekeken worden wat een entiteit zoal kan doen en wat de vervolgacties daarvan zijn. Omdat een entiteit niet zomaar een actie mag uitvoeren zonder een juiste reden.

|  |  |
| --- | --- |
| Naam | *Stealer AI* |

|  |  |
| --- | --- |
| Getest door: | Danny van den Bree |
| Getest op: | PC, Windows, Java |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Test case beschrijving | Verwachte resultaat | Daadwerkelijke resultaat | Opmerkingen |
| 1 | Actie volgen na speler in buurt | De stealer begon met de achtervolging mits de snelheid niet te hoog was. | PASS | geen |
| 2 | Actie pak glyph op | Zodra tijdens seeking een glyph gevonden wordt, werd deze opgepakt | PASS | geen |
| 3 | Geen speler of glyph in de buurt | Gaat over op seeking of naar huis als zijn energy op is. | PASS | geen |
| 4 | Actie steelt glyph van de speler die in de buurt is | Als verwachting ging de seeker in de achtervolging om een glyph af te pakken. | PASS | geen |
| 5 | Genoeg energy | Als er niets te doen is, geen speler of glyph in de buurt, gaat over op seeking. | PASS | geen |
| 6 | Niet genoeg energy | Geen energy, stealer gaat terug naar huis | PASS | geen |
| 7 | Actie thuis | Als de stealer een glyph heeft, dropt deze neer en gaat weer op seeking. | PASS | geen |
| 8 | Glyph in de buurt | Gaat naar glyph en pakt hem op en gaat verder met seeking. | PASS | geen |

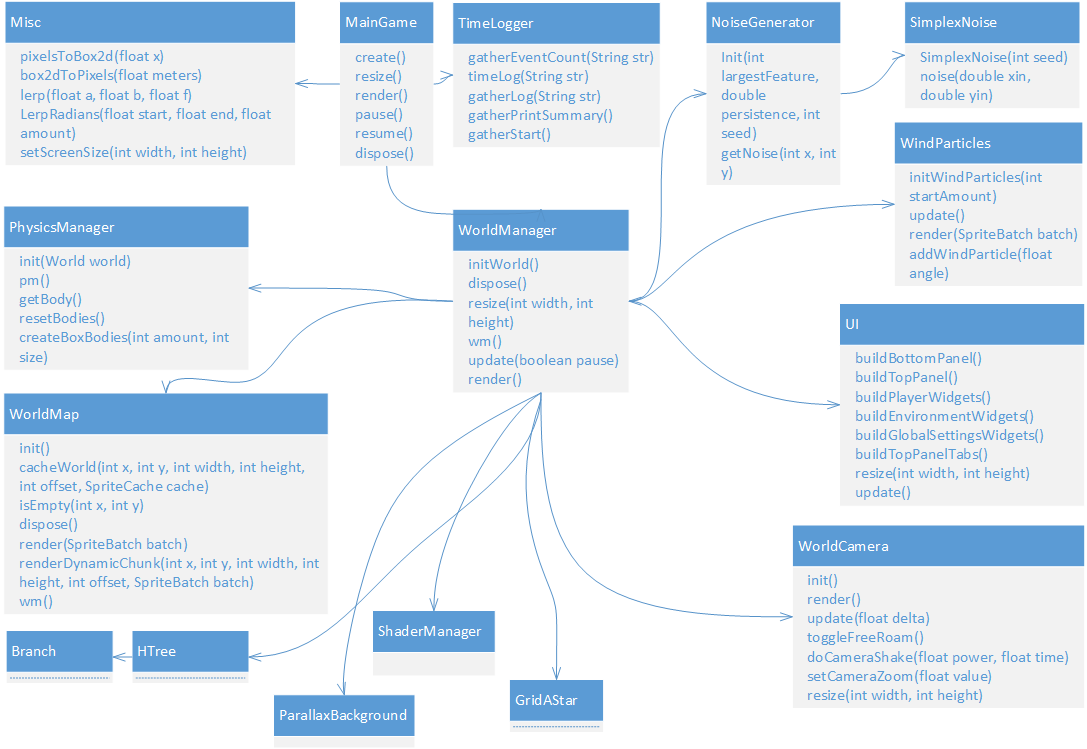
#### Sprint 9 – Path finding (1 mei tot 20 mei)

Als we te maken hebben met AI, kun je niet om pathfinding algoritmes heen, zonder zou elk entiteit hoe intelligent dan ook dom overkomen als het overal tegen aan botst. In plaats van dat ik intelligentie gedragingen ontwikkelen heb ik veel tijd gestoken in pathfinding dat een pad aanneemt dat menselijk overkomt in plaats van altijd de kortste route nemen. Omdat pathfinding in de laatste weken van mijn afstudeerperiode is gebouwd heb ik dit deel kort gehouden.

##### Functioneel ontwerp

Voor deze sprint zijn er een paar klassen bijgekomen. In figuur 38 ziet u een UML klassendiagram met een aantal nieuwe definieerde klassen en deze zal ik in het kort toelichten:

|  |  |
| --- | --- |
| ShaderManager | Deze wordt gebruikt om shaders te renderen |
| GridAStar | Het berekenen van een pad tussen twee punten |



Figuur

##### Technisch ontwerp

Doordat er veel code is geschreven omtrent de pathfinding algoritme kunt u de code terug lezen in de bijlage. Daarnaast is het hele framework een groot project geworden en heb daardoor veel van het functioneel ontwerp achterwege gelaten omdat het meeste werk nog niet af was en geen reden had om het in dit verslag te verwerken. In figuur 39 kunt u zien dat er veel meer werk gedaan is omtrent de entiteiten in combinatie met het Entity component system.

|  |
| --- |
| E:\Dropbox\Screenshots\Screenshot 2014-06-05 04.07.33.png  Figuur |
| Dit is het totaal plaatje waarvan nog veel weggelaten is omdat het nog ontwikkeling was. |

##### Bouw

In figuur 40 ziet u A star in werking, de werking van dit algoritme werkt als volgt. Per positie zoekt het naar de dichtstbijzijnde positie op basis van een heuristiek. Dit betekend dat hij per positie kijkt in welke richting zijn doel staat en kijkt welke posities van zijn huidige locatie beschikbaar zijn die dichterbij zijn. Zodra hij weg van punt A naar B heeft gemaakt moet er een pad uit deze data gefilterd worden. Dit pad wordt berekend door te kijken van alle bezochte posities de kortste route is. Zodra deze route is gemaakt wordt deze opgeslagen en kan worden opgeroepen door één van de entiteiten die de aanvraag heeft gedaan.

|  |
| --- |
| E:\Dropbox\Screenshots\astarpathfinder.png  Figuur |
| A\* in actie, het berekenen van een pad tussen twee punten. |

##### testen

Ik ben er niet aan toegekomen om het goed te testen, het was een soort van experiment om mooiere paden te berekenen. Ik ben vervolgens met de ontwikkeling verder gegaan met de ontwikkeling van de Hence Games waardoor dit alleenstaande project stil werd gelegd.

# Evaluatie

Een afstudeerplan is niet volledig faal veilig als we het hebben over een afwijking in een plan. De hoofdlijnen van het afstudeerplan zijn gevolgd met een aantal zaken die erbij zijn gekomen waar ik aan het begin van mijn afstudeerstage niet aan heb gedacht. Dit heeft te maken dat je in een zakenwereld terechtkomt waarin het niet altijd loopt zoals men wilt. Een voorbeeld hiervan is het stellen van prioriteiten. Deze heb ik vaak genoeg moeten bijstellen om een bepaalde deadlines te kunnen halen. Het is ook niet alleen bij AI gebleven, er is meer bijgekomen dan mijn orginele plan betreft kunstmatige intelligentie. Voor het gehele framework is aardig wat code geschreven dat ook indirect iets te maken had met het ecosysteem. Deze moeten uiteraard ook getest zijn maar omdat deze onderdeel zijn van het LibGDX prototype heb ik deze achterwege gelaten.

## Beantwoording doelstellingen

In hoofdstuk “Opdrachtformulering” heb ik een aantal doelstellingen opgesteld die ik als richtlijn heb gebruikt voor de ontwikkeling van het ecosysteem. Ik zal in deze paragraaf elk van deze doelstellingen doorlopen met daarbij het behaalde resultaat.

Doelstellingen:

1. Er voor zorgen dat de willekeurig geplaatste autonome wezens asynchroon kunnen voortbewegen.
2. Verschillende type entiteiten beschreven, niet alleen wezens maar ook statische entiteiten zoals flora en fauna.
3. Er voor zorgen dat er verschillende type wezens in één wereld aanwezig zijn die ieder anders reageren.
4. Wezens die kunnen veranderen op basis van de elementen die in de omgeving te vinden zijn.
5. Wezens die keuzes kunnen maken op basis van gegeven informatie uit de omgeving.
6. Wezens die doelen kunnen vaststellen.
7. Wezens een geven reden waarom ze in die wereld thuishoren (storyline).
8. Wezens die onverwachte beslissingen kunnen maken.
9. Wezens die op basis van pathfinding hun pad kunnen vastleggen.

Resultaat:

1. Het framework dat ik heb ontwikkeld voor mijn prototype is in staat om verschillende wezens willekeurig in de wereld te plaatsen die ieder zelfstandig op pad gaat.
2. Ik heb een aantal verschillende wezens bedacht waarvan ik één heb geïmplementeerd: Stealer.
3. De twee wezens de Stealer en Charger, ze hebben wel dezelfde set gedragingen.
4. Niet aanwezig door te weinig tijd.
5. De wezens hebben de mogelijkheid om op basis van de map te kunnen navigeren.
6. De wezens kunnen hun doel wijzigingen doorvoeren als zij iets interessant vinden bijvoorbeeld een glyph.
7. Storyline was niet mijn opdracht en heb ik beschouwd als “Nice to have”
8. Wezens kunnen tussendoor keuzes maken, zie punt 6.
9. Wezens kunnen een pad berekenen zodat zij netjes langs entiteiten vliegen en niet overal tegen aan botsen met behulp van A\*.

## Evaluatie opgeleverde producten

Het functionele en technisch ontwerp zijn tijdens iedere sprint van het ecosysteem behandeld omdat een ecosysteem omdat zonder vooronderzoek niet te doen is. Daarom heb ik geen vaste planning aangehouden en ben ik de sprints gaan invullen op basis wat visueel in een korte tijd ontwikkeld kon worden. Het meeste onderzoek heb ik gedaan in het laatste deel van mijn afstudeerperiode met betrekking tot AI, pathfinding en het entity system. Het vooraf schrijven van documentatie in een game development wereld is niet om aan te raden door de vele veranderingen. Het gaat op basis van gebouwde prototypes, ideeën op papier zetten en dat direct vertalen naar code om zo snel mogelijk resultaat te leveren. Het ecosysteem is niet verwerkt in de commerciële build tijdens mijn afstudeerperiode omdat het schrijven van dit verslag eveneens op het programma stond. Deze stap is om die reden weggelaten uit de lijst van sprints omdat dit na mijn afstudeerperiode zal worden uitgevoerd.

Het meeste onderzoek heb ik tijdens het bouwen van het ecosysteem verricht omdat het de ontwikkeling van het framework sterk versnelde maar ook dat er minder werk weggegooid hoeft te worden als er besloten werd om een bepaalde feature uit het spel te schrappen. Daarom heb ik de sprints vrij kort gehouden van maximaal twee tot drie weken.

## Te behalen competenties

In mijn afstudeeropdracht waren er een aantal beroepstaken opgesteld die door mij behaald moesten worden. In deze paragraaf zal ik elk van deze beroepstaken toelichten wat ik qua werkzaamheden heb gedaan om deze beroepstaken te behalen. Voor deze opdracht het ontwikkelen van het Skies of Saturn ecosysteem, komen de volgende beroepstaken aan bod:

|  |  |  |
| --- | --- | --- |
| Nr. beroepstaak | Tekst van de beroepstaak | Beroepstaak genoemd onder punt 6 |
| **1** | Ontwerpen | Functioneel en Technisch ontwerp |
| **2** | Bouwen applicatie | Bouwen |
| **3** | Initiëren en plannen van het testproces | Testplan |
| **4** | Uitvoeren van en rapporteren over het testproces | Testrapport |

1. Ontwerpen
   1. Ik heb voor mijn afstudeeropdracht een ecosysteem ontworpen op basis van de gewenste gameplay elementen die FourceLabs graag in het spel Skies of Saturn geïmplementeerd wilde hebben.
   2. Voor het ontwerpen heb ik met de vooraf beschreven methodes en technieken modellen ontworpen die vervolgens gebruikt werden voor de bouw van een prototype.
2. Bouwen applicatie
   1. Ik heb voor de opdracht een werkend prototype opgeleverd waarin verschillende AI entiteiten keuzes maakt op basis van hun omgeving op basis van een set opgestelde regels/keuzes.
3. Initiëren en plannen van het testproces
   1. Ik heb voor het testproces een aantal testcases opgesteld waarin het ecosysteem getest wordt op de keuzes die gemaakt worden en of deze gefundeerd zijn. Het is alleen gebleven bij een unit test van de entiteit “Stealer”.
4. Uitvoeren van en rapporteren over het testproces
   1. Ik heb voor het testen gekeken naar wat voor keuzes entiteiten maakten, niet zozeer visueel maar intern. Omdat ik het bij een Decision Tree heb gehouden zijn de keuzes niet te complex geworden wegens te veel werk.

## In het vervolg beter of anders zou doen

Tijdens het project ben ik tegen een aantal punten aangelopen dat ik bij een soortgelijk project anders zou doen. Wat ik vooral merkte is dat het oorspronkelijke idee waaruit ik mijn afstudeeropdracht heb opgesteld vrij weinig van over is gebleven door de vele veranderingen. Door deze veranderingen was het heel erg lastig om alles goed in te kunnen plannen waardoor er weleens werk verloren ging. Bijvoorbeeld dat bepaalde features zoals trees spawnen bij het crashen, civilisaties en power ups totaal geschrapt werden. Daarnaast de ontwikkeling van AI, het koste me enorm veel werk waardoor ik niet veel tijd aan verschillende entiteiten kon besteden.

Voordat ik überhaupt aan AI kon werken moest ik eerst een framework opzetten dat noodzakelijk was om werkende prototypes te laten zien, “Zien is geloven”. Mede hierdoor is er weinig gedaan met het testen van entiteiten omdat deze relatief eenvoudig zijn bleven. Ik denk dat de opdracht te ambitieus is geweest voor de tijd die ik er voor heb gekregen. Met dit soort projecten weet je nooit wat je tegen kunt komen, ik heb wel veel kunnen doen maar niet veel tot in relatie met de afstudeeropdracht. Uiteindelijk heb ik veel programmeerwerk gedaan betreft de besturing, performance, level generatie, pathfinding en visuals. Dit heeft tot complexe systemen geleid waar ik veel van heb geleerd, maar enigszins teleurgesteld dat ik niet meer had kunnen doen met kunstmatige intelligentie.

# Bijlage A

## Het spel

Skies of Saturn(SoS) is een game die draait om exploratie. De speler bestuurt een schip dat zich rond beweegt met behulp van de wind. Het schip heeft een zeil dat in en uitgeklapt kan worden. Wanneer het zeil uitgeklapt wordt versnelt het schip, afhankelijk van de windrichting. Wanneer de zeilen ingeklapt zijn kan het schip van de speler onafhankelijk van de windrichting zijn snelheid behouden. Door de wind op de juiste manier te gebruiken kan de speler het schip snel en efficiënt besturen.

Wanneer het spel start wordt de speler een onbekende wereld ingestuurd. Deze wereld wordt procedureel gegenereerd en bestaat uit verschillende biomen(vormen). Tijdens de ontdekkingstochten binnen deze biomen komt de speler allerlei flora en fauna tegen. Door op onderzoek te gaan, te observeren en te proberen kan de speler allerlei interacties met deze entiteiten aangaan.

In de wereld van Skies of Saturn zijn ook restanten van oude beschavingen (landmarks) te vinden: Lang vergeten ruïnes, kapotte machines, mysterieuze artefacten en vreemde locaties. De meest voorkomende artefacten zijn de glyphs. Dit zijn oude artefacten die geladen zijn met een mysterieuze kracht. De speler kan deze glyphs verzamelen en gebruiken om de vele oude machines en ruïnes, die diep in de vele uithoeken en verborgen plekken van Saturn liggen, te activeren.

Wanneer een landmark actief is, kan de speler een unieke speelervaring aangaan met de landmark. De landmark vormt een unieke beleving binnen de wereld van SoS. Wanneer de speler een landmark heeft uitgespeeld, wordt hij beloond met een souvenir van de desbetreffende landmark. Dit souvenir verschijnt in een van de basissen en kan gebruikt worden om de wereld van Skies of Saturn, of de speler te beïnvloeden. Hierdoor kunnen er nieuwe delen van de wereld beschikbaar worden, en kan de speler op nieuwe manieren met de wereld interacteren.

Naast het thema van ontdekking laat Skies of Saturn de speler nadenken over zijn rol in het ecosysteem van de game en over symbiotische/parasitaire verbanden die de speler aangaat met de flora en fauna. Wanneer de speler interacteert met omgeving ontstaat er een symbiose; De speler wordt beïnvloed door de omgeving; Het uiterlijk en de functionaliteiten van het schip veranderen steeds meer. Tegelijkertijd hebben de interactiemomenten en de aanwezigheid van de speler ook invloed op de staat van de wereld.

Om dit concept te kunnen bewaken zijn de volgende principes opgesteld:

* Exploratie: Het hoofddoel van de speler is ontdekking. Iedere vorm van ontdekking moet een positieve beleving zijn.
* Joy of motion: De gameplay en de wind/schip dynamiek moeten speels aanvoelen. Zelfs zonder de spanningsboog van ontdekking moet de besturing leuk en intuïtief en ontspannen aanvoelen.
* Speelduur: Een spelsessie mag ongeveer 5 minuten duren. Alle ontdekkingen binnen deze sessie dragen bij aan de overkoepelende spelervaring. Door vaker te spelen krijgt de speler steeds meer van de wereld te zien.
* Beeldtaal: Er wordt geen tekst gebruikt in de game. Alle menu's worden onderdeel van de spelwereld en bestaan uit visuele elementen. Alle vormgeving is abstract en vervreemdend.

## Gameplay

### Vliegen

In Skies of Saturn bestuurt de speler een zeilschip om de wereld te verkennen. Het schip bestaat uit een zeil en een staart. De speler manipuleert de zeilen om rond te kunnen vliegen door te schakelen tussen twee modi:

* Zeil open: Het schip versnelt naar de topsnelheid van de wind. De camera zoomt verder uit zodat de speler in deze modus meer van zijn omgeving kan zien. Deze modus dient om de speler zo snel mogelijk van punt A naar punt B te reizen.
* Zeil dicht: de camera zoomt in op het schip en het schip beweegt met een snelheid gelijk aan huidige snelheid van het schip. Deze modus dient om de speler nauwkeurig te laten manoeuvreren.

|  |
| --- |
|  |
| Zeil open, Zeil dicht |

### Mobiele platforms

Aan de rechterkant van het scherm is een ring te zien. Door de ring te draaien verandert de wijsrichting van het schip. Wanneer de speler een tweede vinger op het scherm houdt klapt het zeil open. Laat de speler die tweede vinger los dan klapt het zeil weer dicht.

### Muisbesturing

De speler bestuurt het schip met de muis. Het zeil wordt bestuurd met de linker muisbutton.

### Gamepad besturing

De speler bestuurt het schip met de rechter analoge stick. Elke facebutton en schouderbutton kan gebruikt worden om het zeil te besturen.

## Het basis

Het basis is het startpunt en eindpunt van iedere speelsessie van Skies of Saturn en dient als overzichtsscherm van het spel. Vanuit het basisscherm kan de speler een nieuwe verkenningstocht starten.

|  |
| --- |
| basisscherm |
| Voorbeeld van een basis |

Dit doet de speler door te klikken op het schip. Het basisscherm zoomt uit en gaat vervolgens terug naar de wereld map. Hier kan de speler kiezen uit twee vertrekrichtingen. Wanneer de speler een richting heeft gekozen lanceert het schip en begint de verkenningstocht door de gekozen regio.

|  |
| --- |
| fogged |
| Een basis is omgegeven door willekeurige gegenereerde landschappen |

Wanneer het spel voor het eerst gespeeld wordt is er maar één basis beschikbaar. De speler zal de andere twee basissen moeten ontdekken om deze te kunnen gebruiken als nieuw startpunt.

|  |
| --- |
| basissen |
| Er zijn drie verschillende basissen aanwezig |

In totaal zijn er drie verschillende basissen die gebruikt kunnen worden om de wereld van Skies te verkennen. De wereld is onderverdeeld in drie regio’s (levels). Deze levels zijn verbonden door drie basissen. De speler vertrekt altijd vanuit het laatst bezochte basis.

|  |
| --- |
| regios |
| De drie basissen zijn op deze manier met elkaar verbonden |

Als een speler klaar is met zijn verkenningstocht kan hij terugkeren naar het basis waar hij vandaan kwam, of het andere aangrenzende basis betreden.

## De wereld van Skies

De wereld van Skies of Saturn bestaat dus uit drie levels. Ieder level vormt zijn eigen bioom met unieke flora en fauna, ruïnes, grotten, mysteries en uitdagingen. Deze levels worden procedureel gegenereerd en zullen iedere keer een andere lay-out hebben. Op deze manier is er dus altijd iets nieuws te ontdekken voor terugkerende spelers.

De speelduur is ontworpen om in sessies van ongeveer 2 minuten te spelen. Wanneer het spel start, vertrekt de speler vanaf het laatst bezochte basis en kan hij gaan verkennen. Om een idee te geven van de level structuur tussen de verschillende basissen volgt hieronder een voorbeeldafbeelding:

|  |
| --- |
|  |
| Een uitvergroot deel van het level |

De paarse lijnen laten de route tussen twee basissen zien. De paarse stippen vertegenwoordigen de windmarkers. De levels van SoS zijn te verdelen in drie zones. Elke zone is ontworpen met een bepaalde speelduur in gedachte. In het figuur hieronder ziet u een voorbeeld van deze zones uitgebeeld.

|  |
| --- |
| level_zones |
| Route tussen basis A en B |

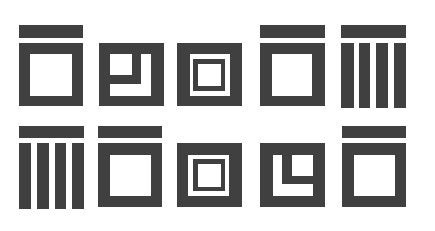
## Windmarkers

De wind speelt een prominente rol in Skies of Saturn. De wind staat nooit stil maar vormt luchtstromen die de speler van punt A naar punt B brengt. Deze luchtstromen waaien altijd in de richting van de wind markers. Wind markers zijn oude artefacten die dienen als wegwijzers tussen twee basissen.

Elk level telt vijf markers. Als de speler een marker bereikt, kan hij deze activeren door er overheen te vliegen. Activeert hij deze, dan draait de wind en zal hij de speler naar de volgende marker blazen. De laatste marker blaast de speler naar het dichtstbijzijnde basis. Dit betekent dat de speler nooit echt kan verdwalen; Door met de windrichting mee te vliegen komt de speler uiteindelijk altijd uit bij een volgend basis.

|  |
| --- |
| wind-marker |
| Interactie met een windmarker, deze kunnen na activatie de windrichting aanpassen. |

## Glyphs

Glyphs zijn oude artefacten die een geheimzinnige kracht bezitten, en zijn de energiebronnen van SoS. Telkens wanneer de speler een level speelt, worden deze glyphs op willekeurige plekken in de wereld geplaatst. Wanneer de speler een glyph vindt, kan hij deze oppakken en gebruiken om landmarks en souvenirs te activeren. Wanneer de speler dat doet worden de glyphs opgebruikt en verdwijnen de glyphs tot de volgende keer dat de speler het level opnieuw betreedt.

### Glyphs oppakken

Wanneer de speler een glyph vindt, kan hij deze meenemen door er overheen te vliegen. De glyph zal de speler gaan volgen. Er zit geen limiet aan de hoeveelheid artefacten die de speler mee kan nemen. Botst een speler, of wordt hij aangevallen door een ander wezen, dan laat de speler al zijn glyphs vallen en zal hij ze snel moeten oppakken voordat de wind ze wegvoert, of andere wezens ze stelen.

### glyph_pickupGlyphs gebruiken

Glyphs kunnen gebruikt worden om landmarks te activeren. Iedere landmark heeft een vaste hoeveelheid artefacten nodig om geactiveerd te kunnen worden. De speler kan de verzamelde glyphs inleveren door dicht langs het apparaat te vliegen. De glyphs worden dan automatisch opgenomen door het apparaat. De speler hoeft niet in één keer de benodigde glyphs in te leveren; Sommige landmarks hebben een enorme hoeveelheid energie nodig om weer op te kunnen starten. De speler zal dan meerdere keren het level moeten spelen om de juiste hoeveelheid glyphs te kunnen verzamelen.

## Souvenirs

Souvenirs zijn speciale pick-ups die uitsluitend verkregen kunnen worden wanneer de speler een landmark heeft uitgespeeld. Heeft de speler dit op de juiste manier gedaan, dan krijgt de speler een souvenir om mee te nemen naar het basis. Deze souvenirs blijven achter in de basissen en kunnen gebruikt worden om de speler, of de spelwereld te beïnvloeden; Wanneer een souvenir in het basis geactiveerd wordt, worden de levels aangrenzend aan het basis beïnvloed. Omdat elk level aan twee basissen grenst, kan ieder level dus door een combinatie van souvenirs beïnvloed worden. Deze combinaties leveren nieuwe belevingen op in het level. Door de grote hoeveelheid van mogelijke combinaties is er altijd iets nieuws te beleven in de wereld van Skies of Saturn.

# Bijlage B

## Tileset

|  |
| --- |
| **private** Array<Vector2> sides = **new** Array<Vector2>();  **private** HashMap<Integer, Sprite> sprites = **new** HashMap<Integer, Sprite>();  **for** (**int** i = 0; i < 16; i++){  *wm*().sprites.put(i, **new** Sprite(**new** Texture(Gdx.*files*.internal("blocks/"+i+".png"))));  }  // i & j zijn de x & y coördinaten van de NoiseGenerator, met een loop alle pixels opvragen binnen het frustum van de camera, alles buiten beeld negeren we.  **final** **double** tile = NoiseGenerator.*getNoise*(i, j);  **if** (tile < -.2f){  **int** score = 0;  **if**(NoiseGenerator.*getNoise*(i, j + 1) < -.2f) score += 1;  **if**(NoiseGenerator.*getNoise*(i - 1, j) < -.2f) score += 8;  **if**(NoiseGenerator.*getNoise*(i + 1, j) < -.2f) score += 2;  **if**(NoiseGenerator.*getNoise*(i, j - 1) < -.2f) score += 4;  **if**(sprites.containsKey(score)){  Sprite sprite = sprites.get(score);  sprite.draw(batch);  }  // zet box2d body verkregen van de physicsManager, zie Physics World Tiles  } |

## Physics

|  |
| --- |
| // het aanmaken van een Box2D body, ik geef deze body sinds het landschap is de categorie LandScape en kan met alles botsen.  **public** **void** **createBoxBodies**(**int** amount, **int** size){  **if**(!initialised) **return**;  **for** (**int** i = 0; i < amount; i++) {  BodyDef bodyDef = **new** BodyDef();  bodyDef.type = BodyType.StaticBody;    PolygonShape shape = **new** PolygonShape();  shape.setAsBox(size/2f, size/2f);    FixtureDef fixDef = **new** FixtureDef();  fixDef.shape = shape;  fixDef.filter.categoryBits = (**short**) PhysicalType.Category.*LandScape*;  fixDef.filter.maskBits = (**short**) PhysicalType.Mask.*All*;    bodyDef.active = **false**;    Body body = world.createBody(bodyDef);  body.createFixture(fixDef);    bodies.put(i, body);  }  }  // in WorldMap wordt de volgende stuk code gebruikt bij het opvragen van een body en te //verplaatsen naar de gegeven lokatie.  **final** Body body = PhysicsManager.*pm*().getBody();  **if** (body == **null**){  **continue**;  }  body.setActive(**true**);  body.setTransform((i \* texScale) + texScale / 2f, (j \* texScale) + texScale / 2f, 0);  body.setUserData(tile); |

## Noisemap

|  |
| --- |
| //NoiseGenerator class  // largestFeature = de grootte van de eilanden  // persistence = de ruwheid, van ronde vormen tot kartelige opgebroken randen.  public static void Init(int largestFeature, double persistence, int seed){  *ng*().largestFeature = largestFeature;  *ng*().persistence = persistence;  *ng*().seed = seed;  //Receives a number (e.g 128) and calculates what power of 2 it is (e.g 2^7)  int numberOfOctaves = (int) Math.*ceil*(Math.*log10*(largestFeature) / Math.*log10*(2));  *ng*().octaves = new SimplexNoise[numberOfOctaves];  *ng*().frequencys = new double[numberOfOctaves];  *ng*().amplitudes = new double[numberOfOctaves];  Random rnd = new Random(seed);  for (int i = 0; i < numberOfOctaves; i++){  *ng*().octaves[i] = new SimplexNoise(rnd.nextInt());  *ng*().frequencys[i] = Math.*pow*(2, i);  *ng*().amplitudes[i] = Math.*pow*(persistence, *ng*().octaves.length - i);  }  }  public static double getNoise(int x, int y){  double result = 0;  for (int i = 0; i < *ng*().octaves.length; i++)  {  result = result + *ng*().octaves[i].noise(x / *ng*().frequencys[i], y / *ng*().frequencys[i]) \* *ng*().amplitudes[i];  }  return result;  } |

## Wind particles

|  |
| --- |
| **class** WindParticle{  **public** **float** x, y, size, scale, dirX, dirY, vel, rot, rotSpeed;  **public** **int** rotDir, particleID;  **public** **float** lifeTime; // if zero, die -> despawn.  }  **public** **void** **update**(){  **if** (windParticles.size() < MAX\_PARTICLES){  addWindParticle(WorldManager.*wm*().getWindAngle());  }  }  **public** **void** **render**(SpriteBatch batch){  **for** (WindParticle obj : windParticles.values()){  **if** (obj == **null**) **continue**;  **float** dirX = MathUtils.*cos*(WorldManager.*wm*().getWindAngle());  **float** dirY = MathUtils.*sin*(WorldManager.*wm*().getWindAngle());  obj.x += dirX \* obj.vel;  obj.y += dirY \* obj.vel;  obj.rot += obj.rotSpeed;  obj.lifeTime -= Gdx.*graphics*.getDeltaTime();  windParticleTex.draw(batch);  **if** (obj.lifeTime <= 0) { // als timer is afgelopen, verwijder particle.  **if** ((obj.a -= Gdx.*graphics*.getDeltaTime()) < 0f){  windParticles.remove(obj.particleID);  **continue**;  }  } **else** **if** (obj.a >= 0f){ // als particle nieuw is, doe dan een fade in  obj.a += Gdx.*graphics*.getDeltaTime();  **if** (obj.a > obj.maxAlpha) obj.a = obj.maxAlpha;  }  } |

## Flyer animation

|  |
| --- |
| public void initFlyerWing(String type){  if(type.isEmpty()) type = "wingbase";    textureAtlas = new TextureAtlas(Gdx.*files*.internal("objects/"+type+".atlas"));  // This loads skeleton JSON data, which is stateless.  SkeletonJson json = new SkeletonJson(textureAtlas);  json.setScale(.01f);  SkeletonData skeletonData = json.readSkeletonData(Gdx.*files*.internal("objects/wing.json"));  // Skeleton holds skeleton state (bone positions, slot attachments, etc).  skeleton = new Skeleton(skeletonData);  skeleton.setX(0);  skeleton.setY(0);  skeleton.setFlipX(true);  // Defines mixing (crossfading) between animations.  AnimationStateData stateData = new AnimationStateData(skeletonData);  animationState = new AnimationState(stateData); // Holds the animation state for a skeleton (current animation, time, etc).  anim1 = stateData.getSkeletonData().findAnimation("anim1"); // animatie uit spine  anim2 = stateData.getSkeletonData().findAnimation("anim2"); // animatie uit spine  }  Render loop…  if (wingClosed) // als linker muis knop actief is  {  if (wingCollapse < 1) { // als animatie niet volledig is afgespeeld  wingCollapse += 0.01f;  }  anim1.mix(skeleton, 0, 1, false, null, wingCollapse); // mix de animatie  } else{  anim2.mix(skeleton, currentAnimationTimer, currentAnimationTimer, false, null, .1f);  wingCollapse = 0;  }  // zet positie/rotatie en render het naar scherm  skeleton.setX(tmp.x);  skeleton.setY(tmp.y);  skeleton.getRootBone().setRotation(-(getRotation() - MathUtils.PI/2) \* MathUtils.radiansToDegrees);  skeleton.updateWorldTransform();    skeletonRenderer.draw(batch, skeleton); |

## Tree generation

Ik heb delen van de code weggelaten om leesbaarheid van de code te behouden. De Branch klasse is verantwoordelijk voor het renderen van de takken en bladeren. De code die ik hierbij heb geschreven gaat als volgt:

|  |
| --- |
| branches = new ArrayList<Branch>();  Branch trunk = new Branch(this, 0, null, posx, posy, startWidth, startHeight, rotation, colorStart);  branches.add(trunk);  public void growTree(){  if(levels == currentLevel) { fullGrown = true; return; }    for (int i = 0; i < branches.size(); i++) {  if(branches.get(i).getLevel() == currentLevel)  {  calculateTree(branches.get(i), currentLevel+1);  branches.get(i).init();  }  }  currentLevel++;  }  private void calculateTree(Branch branch, int level){  if(level >= levels) return;    Vector2 center = branch.getCenterPosition();  float width = branch.getWidth()/(float)Math.sqrt(1.75f);  float height = branch.getHeight()/(float)Math.sqrt(1.75f);  Branch branch\_one = new Branch(this, level, branch, center.x, center.y, width, height, MathUtils.random(branch.getRotation() + branchRotation/2-35, branch.getRotation() + branchRotation/2+5), startColor.cpy().lerp(endColor.cpy(), level/(float)levels));    Branch branch\_two = new Branch(this, level, branch, center.x, center.y, width, height, MathUtils.random(branch.getRotation() - branchRotation/2-5, branch.getRotation() - branchRotation/2+35), startColor.cpy().lerp(endColor.cpy(), level/(float)levels));  branches.add(branch\_one);  branches.add(branch\_two);  } |

## Parallax Background

|  |
| --- |
| **for** (Layer layer : layers.values()){ // doorloop alle lagen  **for** (**int** i = 0; i < layer.objs.size(); i++){ // doorloop alle texture objecten van een laag  BackgroundObject obj = layer.objs.get(i);  obj.lifeTime -= Gdx.*graphics*.getDeltaTime(); // update lifetime, max van ongeveer 1 tot 3 secs    **if** (obj.lifeTime <= 0){ // als lifetime 0 of kleiner is, fade out and delete het vervolgens  obj.currentAlpha = Misc.*lerp*(obj.currentAlpha, 0f, Gdx.*graphics*.getDeltaTime());  **if** (obj.currentAlpha <= .01f){  layer.objs.remove(layer.objs.get(i));  currentInScene--;  **continue**;  }  } **else** **if** (obj.currentAlpha >= 0f){ // als hij nog niet klaar is met in faden, verhoog waarde  obj.currentAlpha = Misc.*lerp*(obj.currentAlpha, layer.alpha, Gdx.*graphics*.getDeltaTime());  }  }  **if**(layer.spriteID == 2){ Farthest background / always show for nice gradient look  **if** (layer.objs.size() < layer.maxObjs) {  layer.objs.add(createBackgroundObject(sprites.get(layer.spriteID), MathUtils.*random*(0, MathUtils.PI2), MathUtils.*random*(layer.minScale, layer.maxScale), MathUtils.*random*(layer.minLifeTime, layer.maxLifeTime)));  }  }  // als er ruimte is voor een nieuwe layer texture plaats hem op een willekeurige positie binnen de camera frustum  **else** **if** (layer.objs.size() < layer.maxObjs && currentInScene <= maxObjectsInScene){  layer.objs.add(createBackgroundObject(sprites.get(layer.spriteID), MathUtils.*random*(0, MathUtils.PI2), MathUtils.*random*(layer.minScale, layer.maxScale), MathUtils.*random*(layer.minLifeTime, layer.maxLifeTime)));  currentInScene++;  }  } Render layers -> layer objects… |

## Vignette Shader

Voor de dynamische achtergrond shader heb ik de volgende GLSL shader geschreven:

|  |
| --- |
| float field(in vec3 p)  {  float strength = 1.9; //9. + .03 \* log(1.e-6 + fract(sin(time) \* 4373.11));  float accum = .5;  float prev = 0.;  float tw = 1.;  for (int i = 0; i < 32; ++i) {  float mag = dot(p, p);  p = abs(p) / mag + vec3(-.2, -.4, -1.3);  float w = exp(-float(i) / 5.);  accum += w \* exp(-strength \* pow(abs(mag - prev), 1.3));  tw += w;  prev = mag;  }  return max(0., 4. \* accum / tw - .4);  }  void main()  {  vec4 texColor = texture2D(u\_texture, vTexCoord);  vec2 uv = (gl\_FragCoord.xy / resolution.xy) - 1.;  vec2 uvs = uv \* resolution.xy / max(resolution.x, resolution.y);  vec3 p = vec3(uvs / 16., 0) + vec3(1., -1.1, 1.);  //p += .2 \* vec3(sin(time / 16.), sin(time / 12.), sin(time / 128.));  p.x += mouse.x;  p.y += mouse.y;  p.z += .9 \* sin(time / 128.);  float t = field(p);  float v = (1. - exp((abs(uv.x) - 1.) \* 1.)) \* (1. - exp((abs(uv.y) - 1.) \* 5.));    texColor.rgb = mix(.3, 1., v) \* vec3(2.4 \* t \* t \* t, 1.5 \* t \* t, t);  gl\_FragColor = texColor;  } |

1. <http://speelbaars.com/?page_id=136> 2014 [↑](#footnote-ref-1)
2. <http://en.wikipedia.org/wiki/Flower_(video_game)> 15 September 2014 at 20:05 [↑](#footnote-ref-2)
3. <http://en.wikipedia.org/wiki/Journey_(2012_video_game)> 18 September 2014 at 04:34 [↑](#footnote-ref-3)
4. <http://www.dutchgamegarden.nl/> [↑](#footnote-ref-4)
5. <http://graylakestudios.com/> [↑](#footnote-ref-5)
6. <http://www.hencegames.com/> [↑](#footnote-ref-6)
7. <http://gamefonds.nl/> 2014 [↑](#footnote-ref-7)
8. <http://en.wikipedia.org/wiki/Procedural_generation> 13 September 2014 at 01:35 [↑](#footnote-ref-8)
9. <http://en.wikipedia.org/wiki/Pathfinding> 19 September 2014 at 06:30 [↑](#footnote-ref-9)
10. <http://libgdx.badlogicgames.com/download.html> versie 0.99 [↑](#footnote-ref-10)
11. <http://nl.wikipedia.org/wiki/Unified_Modeling_Language> 5 mei 2014 om 12:03 [↑](#footnote-ref-11)
12. <http://developer.android.com/sdk/index.html> [↑](#footnote-ref-12)
13. <http://en.wikipedia.org/wiki/Waterfall_model> 30 September 2014 at 16:28 [↑](#footnote-ref-13)
14. <http://en.wikipedia.org/wiki/Rational_Unified_Process> 25 September 2014 at 19:57 [↑](#footnote-ref-14)
15. <http://en.wikipedia.org/wiki/PRINCE2> 18 September 2014 at 22:33 [↑](#footnote-ref-15)
16. <http://en.wikipedia.org/wiki/Scrum_(software_development)> 27 September 2014 at 19:56 [↑](#footnote-ref-16)
17. <http://en.wikipedia.org/wiki/Scrum_(software_development)#Product_backlog> 27 September 2014 at 19:56 [↑](#footnote-ref-17)
18. <http://www.gamasutra.com/blogs/BenWeber/20120308/165151/ABL_versus_Behavior_Trees.php> 03/08/12 06:07:00 pm [↑](#footnote-ref-18)
19. <http://en.wikipedia.org/wiki/Perlin_noise> 24 September 2014 at 08:08 [↑](#footnote-ref-19)
20. <http://en.wikipedia.org/wiki/Simplex_noise>  23 September 2014 at 13:09 [↑](#footnote-ref-20)
21. <http://www.tuicool.com/articles/YjARJf> [↑](#footnote-ref-21)
22. <http://gamedev.stackexchange.com/questions/51693/decision-tree-vs-behavior-tree> [↑](#footnote-ref-22)
23. <http://a-maze.net/> [↑](#footnote-ref-23)