

Entity Behaviour Editor Money Maker Deluxe

Gemaakt in Unity3D



Auteur:	T.Y.P. Vaessen
Type:	Bachelor scriptie
Begeleider:	M.A.P. van Kuijk
Opleiding:	Fontys HBO ICT Eindhoven
Versie:	1.0

AFSTUDEERSCRIPTIE VOOR FONTYS HOGESCHOOL ICT

Gegevens student(e):

Naam: T.Y.P. Vaessen
Studentnummer: 2375907
Hoofdprofiel: Software Engineering
Specialisatie: Game Design & Technology (GD&T)
Afstudeerperiode: 29-08-2015 t/m 03-02-2017

Gegevens bedrijf:

Naam: Firebrush Studios
Adres: Europalaan 20
Postcode: 3526 KS
Plaats: Utrecht
Afdeling: N.v.t

Gegevens bedrijfsbegeleider

Naam: Paul Brinkkemper
Functie: Code Chef
Email: info@firebrushstudios.nl

Gegevens docentbegeleid(st)er:

Naam: M.A.P. van Kuijk

Gegevens afstudeerscriptie:

Titel afstudeerscriptie: Entity Behaviour Editor Money Maker Deluxe
Datum uitgifte: 10-01-2017
Vertrouwelijk: Nee

Getekend voor gezien door bedrijfsbegeleid(st)er:

Datum:

10-1-2016

De bedrijfsbegeleider,
Paul Brinkkemper



Document geschiedenis

Revisies

Versie	Status	Datum	Wijzigingen
0.1	Concept	29-11-2016	Document opzet, hoofdstuk indeling. Bedrijfsbeschrijving en Organogram gemaakt.
0.2	Concept	05-12-2016	Project omschrijving, doelstellingen hoofd- en deelvragen toegevoegd.
0.3	Concept	08-12-2016	Onderzoekshoofdstuk technieken/methodes, Vooronderzoek en interface gemaakt.
0.4	Concept	15-12-2016	Resultaten grondstoffen, components en bedrijfsoverzicht toegevoegd. Project omschrijving beter gedefinieerd.
0.5	Concept	23-12-2016	Conclusies en Reflectie toegevoegd, Resultaten MicroMachinations beschreven.
0.6	Concept	01-01-2017	Woord verminderen, algemene feedback Mark en Riemer verwerken.
1.0	Final	10-01-2017	Spellingscontrole, Feedback op eerdere versie verwerkt.

Distributie

Versie	Datum	Naam	Functie
0.1	05-12-2016	Mark van Kuijk	Docentbegeleider
0.2	09-12-2016	Mark van Kuijk	Docentbegeleider
0.2	09-12-2016	Riemer van Rozen	
0.3	20-12-2016	Mark van Kuijk	Docentbegeleider
0.4	23-12-2016	Paul Brinkkemper	Bedrijfsbegeleider
0.4	23-12-2016	Riemer van Rozen	
0.5	01-01-2017	Mark van Kuijk	Docentbegeleider
1.0	10-01-2017	Mark van Kuijk	Docentbegeleider
1.0	10-01-2017	Paul Brinkkemper	Bedrijfsbegeleider
1.0	10-01-2017	Riemer van Rozen	

Voorwoord

Mijn naam is Tom Vaessen ik heb deze scriptie geschreven tijdens mijn afstudeer stage bij Firebrush Studios te Utrecht. Dit project is uitgevoerd als onderdeel van mijn Software Engineering opleiding met de specialisatie Game Design & Technology aan de Fontys Hogeschool te Eindhoven. Dit project is door mij uitgevoerd tussen augustus 2016 en februari 2017.

Tijdens het project heb ik een systeem ontwikkeld dat het mogelijk maakt gedrag van bedrijven dynamisch aan te passen.

Ik wil graag de volgende personen bedanken voor hun hulp en ondersteuning tijdens het project:

Mark van Kuijk

Paul Brinkkemper

Riemer van Rozen

Stefan Leijnen

Anders Bouwer

Frank Severijns

Samenvatting

Nederlands

Firebrush Studios ontwikkelt Money Maker Deluxe, een economisch bank spel gericht op het creëren van bewustzijn over de huidige problematiek omtrent krediet schepping. De ontwikkeling van Money Maker Deluxe is geïnspireerd door de banken crisis van 2008.

Money Maker Deluxe maakt gebruik van een complex systeem van Regels, Acties en Beperkingen waarmee het gedrag van bedrijven gecontroleerd en bestuurd word. Het aanpassen van deze Regels, Acties en Beperkingen is een langdurig en complex proces. Dit komt door de structuur van het project en de noodzaak om deze aanpassingen volledig in code te moeten realiseren.

Firebrush Studios wilt een systeem ontwikkelen dat het mogelijk maakt de Regels, Acties en Beperkingen dynamisch aan te passen en uit te breiden. Het doel van het systeem is om het gemakkelijker om Regels, Acties en Beperkingen aan te passen en toe te voegen. Verder wilt Firebrush Studios het mogelijk maken om deze aanpassingen mogelijk te maken door een designer.

Om een duidelijk beeld te vormen van de benodigde aanpassingen aan het bestaande project is er gekeken naar de bestaande architectuur en code kwaliteit. Dit is gedaan om verbeterpunten in het project van te voren te identificeren.

Het onderzoek focust zich op het vinden van bestaande technieken en methodes om een dynamisch systeem van Regels, Acties en Beperkingen te realiseren. De belangrijkste technieken en methodes maken gebruik van modularisatie om zo een herbruikbaar en onderhoudbaar systeem op te bouwen.

Voor de ontwikkeling van het project is gekozen voor Entity Component System. Deze techniek staat het toe om een systeem van single responsibility components te maken. Deze components werken onderling samen om het systeem van Regels, Acties en Beperkingen te vormen. Door de modularisatie van Regels, Acties en Beperkingen in kleine onderdelen blijft het project overzichtelijk en onderhoudbaar. Het project maakt gebruik van Editor Windows om Grondstoffen, Bedrijven en bedrijfsgedrag (Regels, Acties en Beperkingen)n te visualiseren en aan te passen.

English

Firebrush Studios is currently developing Money Maker Deluxe which is an economic banking game with the goal of creating awareness regarding credit creation and the problems this creates. The game is inspired by the banking crisis of 2008.

Money Maker Deluxe uses a complex system of Rules, Actions and Conditions to control and govern the behaviour that a company can undertake. Making changes to the system of Rules, Actions and Conditions is a complex and prolonged process because of the complexity of the project and the need to make these changes completely in code.

Firebrush Studios wants to be able to realize changes to Rules, Actions and Conditions by a Game Designer. To be able to do this the existing system needs to be replaced by a dynamic system of Rules, Actions and Conditions. The new system has to enable a Game Designer to change Rules, Actions and Conditions.

To get a clear picture of the changes that needed to be made to the existing system, a technical assessment of the existing system/project was made. This was done by looking at the existing architecture and quality of the code to identify possible strong and weak point within the project.

The research focuses on finding existing techniques and methods which can be used to realize a dynamic system of Rules, Actions and Conditions. The techniques and methods found during this research mainly focus on separation of concerns to create a system with small reusable components. These components are easy to create, reuse and maintain.

The system is created with Entity Component System which was found during the research fase of the project. This technique focuses on creating single responsibility components. Multiple of these components working together create behaviour and the conditions under which this behaviour can be executed. Replacing the existing system of Rules, Actions and Conditions with a modular system of components ensures that the system is easy to maintain en expand. The project uses Editor Windows to enable a Game Designer to make changes to Rules, Actions and Conditions. With this interface changes to Resources, Company's and Company Behaviour can be realized.

Begrippenlijst

Begrip	Betekenis
Single responsibility	Klassen zo ontwikkelen dat ze verantwoordelijk zijn voor 1 onderdeel van een groter probleem. Verschillende single responsibility klassen werken samen om grotere functionaliteit te creëren.
Code tanglement	Code die niet met de core functionaliteit van een klasse te maken heeft toch hierin opnemen. Waardoor klassen meer verantwoordelijkheden krijgen dan de bedoeling is.
Code scattering	Verspreiden van code die met dezelfde functionaliteit te maken hebben over verschillende klassen.
Prefab	Een Unity GameObject die opgeslagen is al Asset. Deze kan veelvuldig in een spel gezet worden waarbij ze allemaal dezelfde functionaliteit/waardes hebben.
Editor Window	Script(s) die gebruikt kunnen worden om custom windows te maken in de Unity Editor.

Inhoud

1. Inleiding	9
2. Het Bedrijf.....	10
2.1. Organogram.....	10
2.2. Money Maker	10
3. Projectomschrijving.....	12
3.1. Money Maker Deluxe	12
3.2. Probleemstelling.....	13
3.3. Doelstellingen.....	13
3.4. Onderzoeksplan.....	14
3.5. Plan van aanpak.....	14
4. Vooronderzoek	16
4.1. Klasse diagram.....	16
4.2. SonarQube.....	16
5. Onderzoek	18
5.1. Machinations	18
5.2. Programming Paradigms	19
5.3. Technische oplossingen.....	21
5.4. Interface	23
5.5. Proof of concepts	24
5.6. Gekozen oplossing.....	27
6. Resultaten.....	30
6.1. Grondstofoverzicht.....	30
6.2. Bedrijfsoverzicht.....	32
6.3. Dynamisch gedrag	34
6.4. Micro Machinations	35
7. Conclusies.....	36
Reflectie.....	37
Literatuurlijst	38
Bijlage I: PID.....	39
Bijlage II: Tips & Tops.....	57
Bijlage III: Klasse diagram systeem.....	60
Bijlage IV: User Requirement Specification.....	72
Bijlage V: Klasse diagram POC ECS	83
Bijlage VI: Klasse diagram POC FSM	84

Bijlage VII: Klasse diagram POC Adapt Tower	85
---	----

1. Inleiding

Firebrush Studios ontwikkelt een digitale en fysieke versie van het economisch bank spel genaamd Money Maker. Het onderzoek wordt uitgevoerd als ondersteuning van de ontwikkeling van de digitale versie Money Maker Deluxe (MMD).

Tijdens de ontwikkel periode van MMD is er door meerdere programmeurs onafhankelijk aan het project gewerkt. Hierdoor is de opbouw van het project complex. Daarnaast is er weinig tot geen documentatie beschikbaar. Dit zorgt ervoor dat het maken van relatief eenvoudige aanpassingen aan het gedrag van In-game bedrijf een moeilijk en langdurig proces is.

Het onderzoek draait om het aanpasbaar maken van de bestaande Regels, Acties en Beperkingen waar bedrijven binnen MMD zich aan moeten houden. Dit heeft als doel het aanpassen van het gedrag dat bedrijven kunnen ondernemen en onder welke voorwaarden zij deze acties kunnen ondernemen.

Door het aanpassen van Regels, Acties en Beperkingen mogelijk te maken door een designer moet het ontwikkel proces van MMD versnelt worden. Hierbij is het doel om aanpassingen te kunnen realiseren door een designer zonder hiervoor te hoeven programmeren.

Het onderzoek dient niet alleen als ondersteuning van MMD maar dient ook als casestudy voor het “Live Intelligent Visual Environments for Game Design” onderzoek. Dit onderzoek loopt van maart 2016 tot februari 2018. Het doel van de casestudy is het vaststellen van welke mogelijk tooling voor live game design in het bedrijfsleven nodig is.

In deze scriptie zijn de hier onderstaande hoofdstukken te vinden

Hoofdstuk 2 Het bedrijf: In dit hoofdstuk wordt het bedrijf en de werkzaamheden van het bedrijf beschreven.

Hoofdstuk 3 Project omschrijving: In dit hoofdstuk wordt het project gedefinieerd, hierin staan de probleemstelling, doelstellingen en de hoofd en deelvragen van het project.

Hoofdstuk 4 Vooronderzoek: In dit hoofdstuk wordt de beginsituatie van het project beschreven.

Hoofdstuk 5 Onderzoek: In dit hoofdstuk worden de gevonden technieken en methodes beschreven. Verder worden hier de uitgevoerde acties tijdens het onderzoek beschreven.

Hoofdstuk 6 Resultaten: In dit hoofdstuk is de ontwikkeling van het project en de uitgevoerde iteraties beschreven.

Hoofdstuk 7 Conclusie: In dit hoofdstuk staan de conclusies en bevindingen van het uitgevoerde onderzoek.

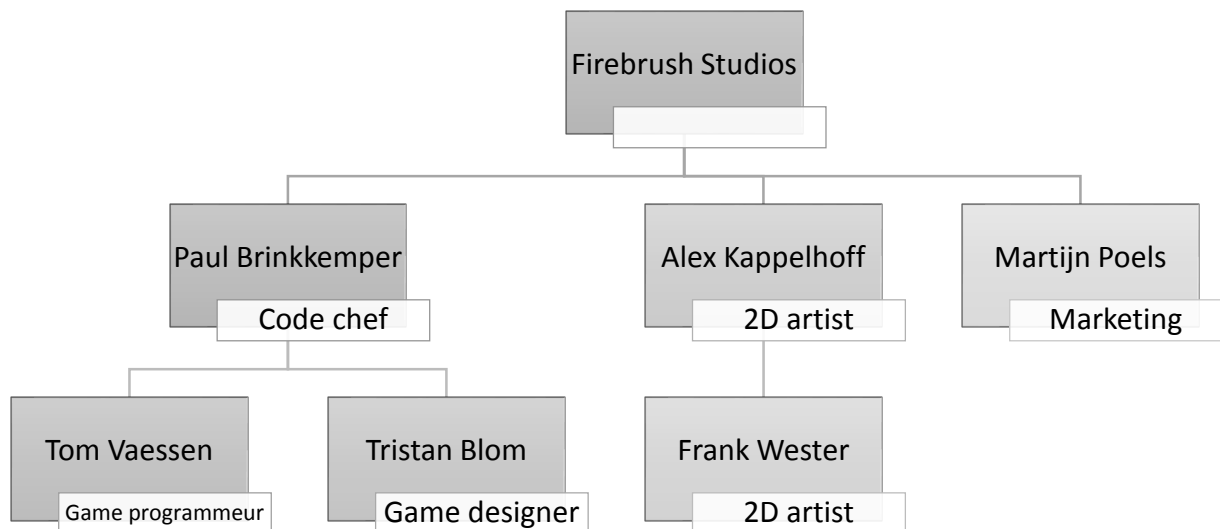
Hoofdstuk Reflectie: In dit hoofdstuk wordt er gereflecteerd op mijn handelingen en professionele houding tijdens de uitvoering van het project.

2. Het Bedrijf

Firebrush Studios is opgericht in 2012 door de 3 vaste medewerkers, met als doel het ontwikkelen van games en aanverwante producten. Het gaat hierbij om zowel digitale als fysieke producten. Naast de ontwikkeling van eigen producten werkt Firebrush Studios ook aan producten voor derden.

2.1. Organogram

In Figuur 1 is de bedrijfsstructuur van Firebrush Studios afgebeeld. Bij Firebrush Studios werken momenteel 3 fulltime vaste medewerkers en 3 fulltime stagiaires.



Figuur 1: Organogram Firebrush Studios

2.2. Money Maker

Firebrush Studios ontwikkelt momenteel zowel een digitale (MMD) als fysieke (Money Maker Moguls) versie van Money Maker. Het draait voor beide versies om het creëren van bewustzijn onder jongeren hoe het huidige financiële stelsel ontstaan is en werkt. Dit is geïnspireerd door de bankencrises van 2008 en de huidige problematiek met betrekking tot krediet schepping.

Money Maker probeert de volgende leerdoelen over te brengen aan de speler(s):

- Bank werking
- Verschil geld & krediet
- Krediet schepping -> inflatie
- Krediet schepping -> bubbel
- Bubbel -> instabiliteit
- Instabiliteit -> Minski moment
- Jubeljaar
- Wijsheid met betrekking tot geld/schulden
- Gouden standaard

In Money Maker speel je als een bank in de Nederlandse Gouden eeuw met als doel de machtigste bank te worden.

Voor de ontwikkeling van Money Maker Moguls(bordspel) is Firebrush Studios een samenwerking aangegaan met stichting Ons Geld. Het fysieke spel is een versimpelde versie van het digitale spel en wordt als losstaand project ontwikkeld.

Mijn project wordt uitgevoerd als onderdeel van het digitale spel. Verdere uitleg over de werking van MMD is te vinden in [Hoofdstuk 3.1](#).

3. Projectomschrijving

In dit hoofdstuk worden de probleemstelling, doelstellingen, plan van aanpak en de hoofd en deelvragen beschreven. Ook wordt in dit hoofdstuk de werking van MMD beschreven. Verdere informatie over het project is te vinden in [Bijlage I: PID](#).

3.1. Money Maker Deluxe

In MMD speel je als een bank in de Nederlandse Gouden eeuw. Hierbij is je doel het vergaren van zo veel mogelijk rijkdom. Om dit te bereiken kan de speler geld lenen en uitlenen aan bedrijven. Door op het juiste moment in de juiste economische sector te investeren kan de speler winst maken of juist veel verliezen.

MMD heeft als begin datum 1 januari 1492. MMD is een turn based spel waarin iedere beurt 1 maand is. Voor een beurt wordt voor iedere dag in de specifieke maand de markt werking en bedrijfsacties gesimuleerd. De eerste maand wordt zonder invloed (en acties) van de speler direct uitgevoerd. Dit wordt gedaan zodat alle bedrijven bestaande handelscontracten hebben en er al een actieve markt werking is.

De spelwereld is opgedeeld in verschillende “dorpen” ieder dorp heeft een eigen lokale markt die gebruikt wordt om grondstoffen te verhandelen. De markten bepalen de prijs van grondstoffen aan de hand van vraag en aanbod binnen zijn respectieve gebied. Een bedrijf kan alleen handelen met zijn lokale markt.

Alle bedrijven binnen MMD hebben als doel het produceren van een of meer grondstoffen en deze te verkopen om zo winst te maken. Om dit te kunnen doen heeft ieder bedrijf een of meer andere grondstoffen nodig die de productie mogelijk maken.

MMD bevat de volgende grondstoffen:

- None
- Wheat
- Credit
- Labour
- Bread
- Tool
- Ore
- Wood

MMD bevat de volgende bedrijven:

- None
- House
- Mill
- Farm
- Smith
- Mine
- Woodcutter
- Trader
- Player
- Bank

De Regels, Acties en Beperkingen waar bedrijven zich aan moeten houden zijn te vinden in [Bijlage VI: User Requirement Specification](#).

3.2. Probleemstelling

Bedrijven binnen MMD werken volgens een systeem van regels. Deze bepalen wanneer en onder welke voorwaarden de bedrijven acties mogen ondernemen. Hierbij gaat het om de acties die een bedrijf binnen de economie kan doen.

Aanpassingen aan regels, acties en beperkingen aan bedrijven binnen het spel MMD vergen veel ontwikkeltijd. Dit wordt voornamelijk veroorzaakt door complexiteit en grootte van het project.

Deze complexiteit is ontstaan doordat er veel verschillende ontwikkelaars onafhankelijk van elkaar aan het project gewerkt hebben. Daarnaast is er geen of weinig documentatie geschreven door de ontwikkelaars wat ertoe geleid heeft dat er verschillende ontwerpen/oplossingen door elkaar gebruikt zijn waardoor het project onoverzichtelijk en gefragmenteerd is.

Aanpassingen aan regels, acties en beperkingen van entiteiten binnen het spel moeten volledig binnen de code gerealiseerd worden. Hierdoor kunnen dit soort aanpassingen bijna niet door een designer gedaan worden.

Door de lange tijd die nodig is voor het realiseren van deze aanpassingen ligt de iteratie tijd van het project hoog. Dit maakt het toevoegen en testen van nieuwe regels een langdurig en inefficiënt proces. Hierdoor duurt het valideren van aanpassingen aan regels of nieuwe regels lang. Waardoor het lang duurt om te valideren of nieuwe regels iets toevoegen aan het spel en de gestelde Educatieve doelen.

Doordat het systeem van Regels, Acties en Beperkingen complex is en het valideren van toevoegingen (of aanpassingen) lang duurt, is het moeilijk om gelijktijdig verschillende nieuwe regels/aanpassingen te realiseren. Dit omdat de alle Regels, Acties en Beperkingen invloed op elkaar hebben en het behaalde resultaat (van entiteiten) beïnvloeden.

3.3. Doelstellingen

Het onderzoek heeft als doelstelling een systeem te ontwikkelen dat het mogelijk maakt Regels, Acties en Beperkingen van entiteiten binnen MMD dynamisch aan te passen. Dit moet gedaan kunnen worden door een Game Designer zonder hiervoor dingen in de code te hoeven aanpassen.

Het onderzoek wordt uitgevoerd als case studie van het “Live Intelligent Visual Environments for Game Design” Raak MKB onderzoek. Dit onderzoek loopt na mijn stage nog +- 2 jaar door. Na de afronding van het Raak MKB onderzoek is de ontwikkeltijd van MMD nog +- 1 jaar.

Verder heeft het project als secundair doel het wegnemen van complexiteit binnen het project en specifiek de code opbouw.

3.4. Onderzoeksplan

In dit onderdeel worden de Hoofd en Deelvragen van het onderzoek gedefinieerd. Deze zijn opgesteld aan de hand van de Doelstellingen van het project en de gestelde eisen en wensen van de opdrachtgever.

3.4.1. Hoofdvraag

Het onderzoek heeft als doel de volgende vraag te beantwoorden:

Hoe kan een systeem ontwikkeld worden dat het mogelijk maakt om regels, acties en beperkingen van entiteiten binnen het spel MoneyMaker Deluxe aan te passen door een designer?

3.4.2. Deelvragen

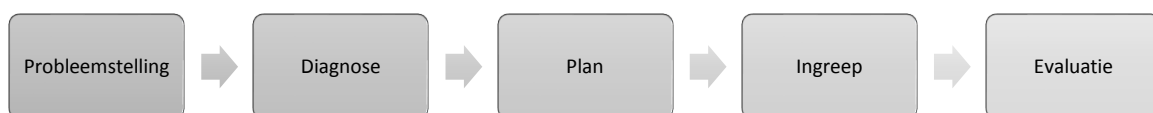
Om de gestelde hoofdvraag te kunnen beantwoorden moeten deze deelvragen beantwoord worden tijdens het onderzoek.

- *Hoe werkt het huidige systeem van regels, acties en beperkingen*
- *Welke eisen worden er aan het systeem gesteld vanuit het bedrijf (designer)*
- *Welke technieken en/of methodes bestaan er om regels, acties en beperkingen van entiteiten dynamisch aanpasbaar te maken*
- *Welke aanpassingen moeten doorgevoerd worden om dit systeem te realiseren*

3.5. Plan van aanpak

Het project wordt uitgevoerd aan de hand van de stappen die in Figuur 2 staan. Deze stappen worden ook benut tijdens de iteratie(s) die gebruikt worden tijdens de implementatie fase. Over de planning van het project is meer informatie te vinden in [Bijlage I: PID](#).

Tijdens het onderzoek wordt gebruik gemaakt van het triangulatie framework. In de hieronder beschreven fases wordt beschreven welke onderdelen van het triangulatie framework gebruikt worden.



Figuur 2: De fases van het project

Probleemstelling

In deze fase werden de eisen en wensen aan het systeem geformuleerd in overleg met de opdrachtgever. Hiervoor werd gebruik gemaakt van Veldonderzoek (interview/gesprekken met de opdrachtgever en raak onderzoekers). Verder is er in deze fase gekeken naar het bestaande systeem en de benodigde aanpassingen om de nieuwe functionaliteit te kunnen creëren.

Diagnose

In deze fase werd doormiddel van Biebonderzoek (literatuuronderzoek) en waar mogelijk Showroomonderzoek (bestaande oplossingen/technieken zoeken) gezocht naar mogelijk oplossingen om de doelstellingen te behalen. Waar nodig is gebruik gemaakt van Werkplaatsonderzoek om gevonden oplossingen te testen en verifiëren.

Plan

Deze fase werd gebruikt om een software ontwerp te maken waarmee de eerder opgestelde doelstellingen gerealiseerd konden worden.

Ingreep

In deze fase werd het eerder gerealiseerde ontwerp gebruikt om een nieuw prototype te ontwikkelen(of bestaand prototype uit te breiden).

Evaluatie

In deze fase werd in overleg met de opdrachtgever het ontwikkelde prototype geëvalueerd. In deze fase zijn verbeterpunten en de vervolg stappen gedefinieerd.

4. Vooronderzoek

In dit hoofdstuk worden de stappen die als voorbereiding van het onderzoek ondernomen zijn beschreven. Deze stappen zijn ondernomen om een duidelijk beeld te vormen van het bestaande project. Het doel hiervan is het bepalen van de technische staat van het project en welke aanpassingen noodzakelijk zijn voor het behalen van de project doelstellingen.

Hiervoor is een Klasse diagram van het bestaande project gemaakt. Verder is er een code analyse gemaakt van het bestaande project doormiddel van SonarQube.

4.1. Klasse diagram

De documentatie die momenteel bestaat is verouderd en incompleet. Hierdoor is er geen duidelijk overzicht hoe de verschillende onderdelen met elkaar samen werken.

Ik heb ervoor gekozen het klasse diagram zelf te maken in tegenstelling tot automatisch laten genereren. Hiervoor is gekozen zodat tijdens het maken van dit klasse diagram door de code gelopen moet worden. Dit bied inzicht in de opbouw en verantwoordelijkheden van de diverse klassen. Het gaat hier om inzicht over de precieze werking van het economische systeem. Verder is hiervoor gekozen omdat de mogelijkheden voor het automatisch genereren van een klasse diagram erg beperkt zijn. De klasse diagram is te vinden in [Bijlage III: Klasse diagram systeem](#).

Naast de klassen die opgenomen zijn in het klasse diagram zijn er +- 70 niet gedocumenteerde klassen. Het draait hier voornamelijk om “single responsibility” klassen. Deze zijn verantwoordelijk voor UI, Navigatie, Wereldgeneratie en Animaties. Deze klassen zijn niet opgenomen omdat deze geen (directe) invloed hebben op het economische systeem.

Het huidige project is erg onoverzichtelijk dit wordt veroorzaakt door onnodige verwijzingen naar klassen en instanties. Ook heeft dit te maken met de huidige indeling van de scripts (weinig onderverdeling). Verder bevat het project veel deprecated en legacy code en worden enums globaal in klassen gedeclareerd waardoor deze niet gemakkelijk terug te vinden zijn.

4.2. SonarQube

Omdat er betrekkelijk veel problemen zijn met betrekking tot het technische ontwerp van het bestaande project heb ik ervoor gekozen doormiddel van SonarQube een code analyse uit te voeren. Hiermee kan de kwaliteit van de code beoordeeld worden. Dit heeft als doel verbeterpunten voor het nieuwe systeem te formuleren alsmede te bepalen of het bestaande project levensvatbaar is.

Het project bevat 34000 regels code waarvan er 12000 direct te maken hebben met de economie. De overige code heeft voornamelijk te maken met aanverwanten systemen onderverdeeld in: Markt, Map, Save/Load en UI.

Met het beoordelen van de kwaliteit van het project is voornamelijk gekeken naar de Bugs, Technical debt, Maintainability en Duplication van het project.

Maintainability

De Maintainability waarde wordt gebaseerd op “code smells” dit zijn alle dingen binnen de code die voor waarschuwingen kunnen zorgen.

Bugs

Bugs zijn gebaseerd op “code smells” maar zouden voor errors kunnen zorgen, hieronder vallen ook beveiligingsproblemen.

Duplication

Hoeveel procent van de code dubbele code/functionaliiteit is. Dit geeft aan dat bepaalde klassen/methoden opgeschoond kunnen worden.

Technical debt

Staat voor de hoeveelheid werk/tijd het kost om de 3 bovenstaande categorieën compleet te verhelpen. De totale hoeveelheid tijd wordt gebaseerd op een gemiddelde waarde per “code smells”. Deze waarde wordt door SonarQube bepaald en is gebaseerd op de severity en type van de “code smell”. Door de totale benodigde tijd te baseren op een gemiddelde is deze waarde over het algemeen correct.

Er is voor deze waarden gekozen omdat deze een goed inzicht in de technische staat van het project bieden. De waarden hiervoor zijn te vinden in Tabel 1.

Naam	Waarde	Rating
Bugs	161	D
Technical debt	28 dagen	N/A
Maintainability	2535	A
Duplication	4.3%	N/A

Tabel 1: waarden SonarQube

Hierbij is het wel belangrijk er rekening mee te houden dat de regels set die SonarQube gebruikt niet voor Unity3D bedoeld is maar voor C#. Hierdoor zitten er +- 600 waarschuwingen bij die direct te maken hebben met hoe de code opgebouwd is binnen Unity. Deze waarschuwingen worden daarom niet meegenomen in de beoordeling.

5. Onderzoek

In dit hoofdstuk worden alle gevonden technieken en methodes beschreven die gebruikt kunnen worden om het project te realiseren. Uit de hier genoemde technieken en methodes worden een of meer technieken/methodes gekozen om het systeem te realiseren. Dit hoofdstuk heeft als doel de gevonden technieken en methodes te beschrijven en de uiteindelijke keus voor een techniek of methode te onderbouwen. Als startpunt van het onderzoek is Machinations genomen, deze techniek wordt beschreven in Hoofdstuk 5.1.

Voor het onderzoek is gebruik gemaakt van de Bieb en Showroom methodiek om bestaande oplossingen en technieken/methodes te vinden.

Voor de Bieb methode is gebruik gemaakt van internet om doormiddel van google en stackoverflow bestaande oplossingen te vinden.

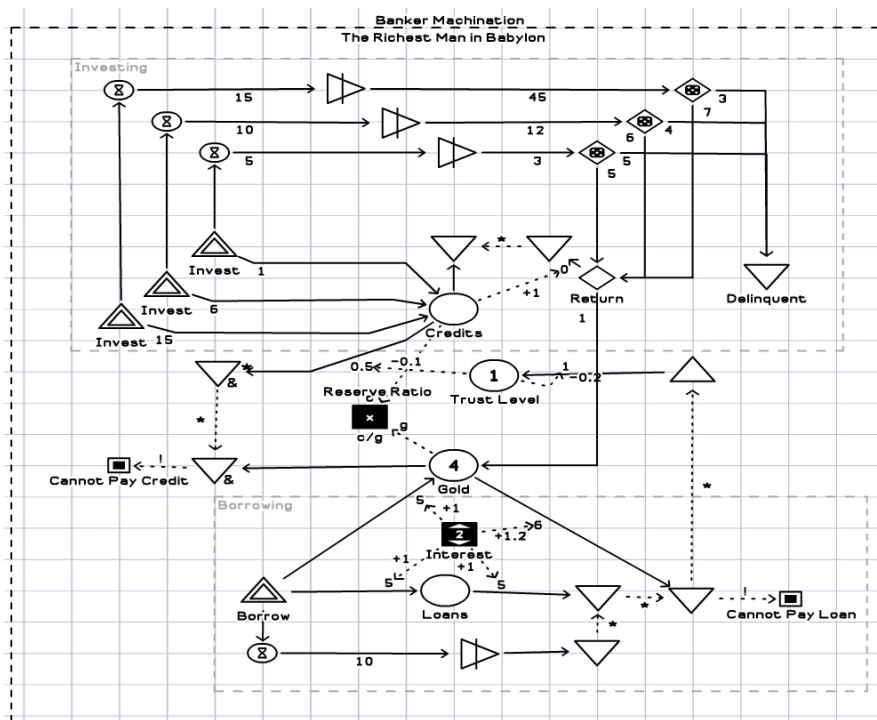
Voor de Showroom methode is gebruik gemaakt van contacten bij bedrijven (via de opdrachtgever en raak onderzoekers) om bestaande vergelijkbare projecten te zoeken.

Naast bovenstaande zoektechnieken zijn er in overleg met de Raak onderzoekers mogelijk aanknopingspunten (en te onderzoeken technieken) bepaald.

Voor alle gevonden technieken en methodes worden aan de hand van het onderzoek voor en nadelen geïdentificeerd. Deze voor en nadelen zijn bepaald aan de hand van de gevonden literatuur over de betreffende techniek of methode.

5.1. Machinations

Machinations wordt in het artikel “Engineering Emergence Applied Theory for Game Design” door Joris Dormans beschreven als ondersteunende game design tool. Machinations maakt het mogelijk economieën binnen een game te visualiseren doormiddel van diagrammen zoals te zien in Figuur 3.



Figuur 3: voorbeeld van Machinations diagram

Deze diagrammen maken het mogelijk om complexe systemen overzichtelijk schematisch weer te geven. Deze diagrammen kunnen gebruikt worden om game mechanics te testen voordat de functionaliteit ontwikkeld wordt.

Machinations biedt 2 mogelijkheden om diagrammen te testen. De eerste optie voert alle genomen acties van de gebruiker direct uit, hierdoor zijn resultaten van genomen acties direct zichtbaar. De tweede optie voert de diagram zelfstandig uit voor een bepaald aantal runs. Hierbij krijgt de gebruiker te zien hoeveel runs succesvol afgerond worden en hoe vaak bepaalde exit condities gebruikt worden.

Deze “simulaties” kunnen gebruikt worden voor het testen en balanceren van de gemodelleerde diagrammen.

Machinations is niet geschikt als techniek voor het te realiseren onderzoeksproject omdat er geen mogelijkheid bestaat de schematische diagrammen in een project te gebruiken.

5.2. Programming Paradigms

In dit onderdeel worden de gevonden technieken en methodes beschreven die beschouwt kunnen worden als Programming Paradigms. De Programming Paradigms zijn geselecteerd op hun mogelijkheden om de bestaande problematiek van het project op te lossen (of tegen te gaan). Het gaat hierbij om de complexiteit binnen de code die veroorzaakt wordt door super classes en weinig of geen separation of concerns. Hierbij is er ook rekening mee gehouden dat de Programming Paradigms geschikt moeten zijn om een dynamisch systeem van Regels, Acties en Beperkingen op te zetten.

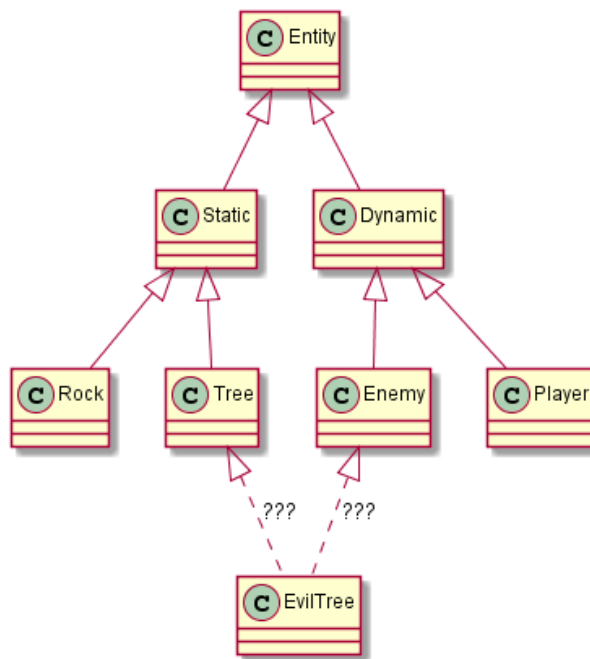
De gevonden Programming Paradigms die aan deze eisen voldoen worden hieronder beschreven.

5.2.1. Component Based Development

Volgens het artikel "A Model of Component-Based Programming" van Douglas McIlroy (1986) draait Component Based Development om het scheiden van verantwoordelijkheden. Ieder component wordt opgebouwd met 1 functie. Hierdoor krijg je kleine components die gebruik maken van “Loosely coupled architecture” waardoor de components beter herbruikbaar zijn. Hierdoor kunnen de betreffende components gemakkelijker (her)gebruikt worden om nieuwe bedrijven en/of bedrijfsgedrag te creëren.

Entity Component System

Entity Component Systems(ECS) zoals die beschreven worden in de blog “Entity Systems are the future of MMOG development” door Adam Martin is een vorm van Component Based Development. ECS deelt de code op in de 3 onderdelen zoals deze te vinden zijn in Figuur 4.



Figuur 4: Schematische weergave ECS uit "Understanding Component-Entity-Systems"

Entity

Een Entity is een instantie binnen de spel wereld.

Component

Components definiëren de Entity en de acties die de Entity kan ondernemen.

System

Het System voert de acties uit die een specifieke Entity kan doen.

De genoemde onderdelen staan het toe gemakkelijk nieuwe entiteiten samen te stellen uit het gedrag van eerder ontwikkelde entiteiten. Ook zorgt deze opbouw ervoor dat het gemakkelijker wordt om bestaande components aan te passen/vervangen doordat deze components geen of weinig afhankelijkheden hebben.

Voor- en Nadelen

- + Components kunnen hergebruikt worden
- + Entiteiten zijn makkelijk uit te breiden
- + Makkelijk te onderhouden
- + Code is overzichtelijk
- Slecht compatible met bestaande architectuur

Aspect Oriented Programming

Aspect Oriented Programming(AOP) zoals het hier beschreven wordt: "An Introduction to Aspect-Oriented Programming and AspectJ" is een vorm van Component Based Development. AOP heeft als doel het tegen gaan van complexiteit binnen een project. Hierbij gaat het vooral om het tegen gaan van het verspreiden van code over verschillende klassen (code scattering). Verder richt AOP zich erop om code te scheiden op functionaliteit.

AOP werkt doormiddel van zo genoemde Aspecten. Een aspect binnen het project is alle code die met een specifiek belang te maken heeft. Hierbij gaat het om belangen die door het hele (of groot gedeelte van) het project gebruikt dient te worden. Denk hierbij aan Logging, Security en Localization.

Het gebruik van Aspecten hiervoor zorgt ervoor dat de benodigde code voor deze belangen op een plaats staat en aangeroepen wordt. Dit voorkomt dat deze code verspreidt wordt over meerdere klassen ("code scattering") en zorgt ervoor dat de verschillende aspecten van elkaar gescheiden blijven ("code tanglement"). Het tegen gaan van deze complexiteit binnen het project zorgt ervoor dat dit overzichtelijk en daardoor onderhoudbaar blijft.

Voor- en Nadelen

- + Werkt goed met objecten/eisen die project breed gebruikt moeten kunnen worden
- Niet compatible met de bestaande architectuur

5.2.2. Model View Controller

Model View Controller(MVC) zoals het beschreven wordt in de blog "Understanding Model-View-Controller" deelt het project/code op in de volgende 3 onderdelen.

Model

Logica en data van het specifieke onderdeel.

View

Weergeven van output.

Controller

Reageert op input van de gebruiker en zet dit om naar commando's voor de View of Model.

Het scheiden van model, view en controller zorgt er voor dat de opbouw van het project overzichtelijk blijft.

Naast MVC zijn er een aantal vergelijkbare architectuur types die in een zelfde manier gebruikt kunnen worden. Hierover is meer informatie te vinden in het artikel "MVVM vs MVP vs MVC: The differences explained".

Voor- en Nadelen

- + Overzichtelijk door scheiding van code(model, view, controller)
- + Kan ook gebruikt worden in combinatie met andere technieken
- + Mogelijkheden voor live inladen gegevens(data files)

5.3. Technische oplossingen

In dit onderdeel worden de overige technieken en methodes beschreven. Deze Technische oplossingen zijn geselecteerd aan de hand van hun geschiktheid om het systeem van Regels, Acties en Beperkingen dynamisch te realiseren.

De gevonden Technische oplossingen die voldoen aan de gestelde eisen worden hieronder beschreven.

5.3.1. Micro Machinations

Micro Machinations(MM) zoals deze beschreven wordt in het artikel “A Pattern-Based Game Mechanics Design Assistant” door Riemer van Rozen is een op zichzelf staande doorontwikkeling van Machinations.

MM werkt met een systeem van diagrammen die runtime ingeladen kunnen worden door MM. Deze diagrammen bestaan momenteel uit Micro Machinations Code in een later stadium is het de bedoeling hiervoor MM Diagrammen voor gebruiken.

Het runtime inladen van een “diagram” staat het toe om regels live aan te passen of nieuwe toe te voegen. Live wilt zeggen dat dit gedaan kan worden als het spel draait. Dit zorgt ervoor dat (Micro) Machinations diagrammen gebruikt kunnen worden als Game Development Tool waarmee complexe economische systemen eenvoudig en overzichtelijk gemodelleerd en gebruikt kunnen worden.

MM ondersteunt het simuleren van diagrammen op de zelfde manier als Machinations (zie Hoofdstuk 5.1) zodat waar nodig diagrammen getest kunnen worden.

Voor- en Nadelen

- + Live aanpassingen mogelijk
- + Diagrammen gemakkelijk testen (doormiddel van simulaties)
- Kan momenteel niet aan Unity gekoppeld worden

5.3.2. Finite State Machines

Finite State Machines(FSM) zoals die beschreven worden in de blog “Finite-State Machines: Theory and Implementation” kunnen gebruikt worden om acties te definiëren en controleren.

Een FSM koppelt acties aan een “state”. In een FSM kan één state tegelijk actief zijn, deze actieve state bepaald de huidige acties die een entiteit kan ondernemen.

Een state kan meerdere transities hebben naar andere states, deze transities zorgen ervoor dat de actieve state kan veranderen. De FSM beheert de transities tussen verschillende states, iedere transitie kan een of meer condities hebben. Een transitie moet voldoen aan alle condities om door te kunnen gaan naar een volgende state. Dit systeem zorgt ervoor dat een entiteit de acties die uitgevoerd worden aanpast aan de omstandigheden binnen het spel

Voor- en Nadelen

- + Goed voor AI
- + Makkelijk entiteit regels, acties en beperkingen aanpassen
- + Volgorde van acties is gemakkelijk te bepalen
- Lastig in gebruik voor spelregels

5.3.3. Visual Programming Tools

Visual Programming Tools(VPT) geven de gebruiker de mogelijkheid code te maken door middel van een interface. Hierdoor kan code gemakkelijker door designers aangepast worden.

Veel VPT bieden de gebruiker een library van veel gebruikte functionaliteiten aan, deze voorbeelden werken out of the box en kunnen waar nodig door de gebruiker aangepast worden.

Een aantal van deze systemen bieden ook de mogelijkheid om bestaande code om te zetten naar een visuele representatie hiervan(Nottorus, uScript).

Voorbeelden van VPT zijn:

Unreal Engine Blueprints

Nottorus ([link](#))
uScript ([link](#))
Playmaker ([link](#))
GameFlow ([link](#))

Voor- en Nadelen

- + Complexe systemen simpel doormiddel van interface in elkaar te zetten
- Niet compatible met het bestaande project

5.4. Interface

In dit onderdeel worden gevonden technieken en methodes voor de interface van het systeem beschreven. Hiernaar is gekeken omdat het systeem aanpassingen aan Regels, Acties en Beperkingen door een designer mogelijk moet maken. Deze aanpassingen zijn voor een designer het gemakkelijkste te realiseren doormiddel van een interface (of vergelijkbare oplossing).

Hieronder worden de verschillende mogelijkheden voor de interface beschreven.

5.4.1. Inspector

Inspector values maken het mogelijk om variabelen van een klasse aan te passen binnen een Unity scene.

Het huidige systeem maakt hiervan gebruik om een aantal waardes van bedrijven en andere objecten aan te passen. Dit moet voor het project op de specifieke prefab gebeuren. Hierdoor is het erg moeilijk om overzicht te houden van de verschillende bedrijven en de waardes hiervan.

5.4.2. Editor Windows

Editor Windows bieden de mogelijkheid om Unity tools te ontwikkelen als ondersteuning van het project.

5.4.3. Node Editors

Unity3D biedt standaard de mogelijkheid om met behulp van een Editor Window een Node Editor te maken. Een Node Editor maakt het mogelijk gemakkelijk en overzichtelijk de flow binnen een onderdeel aan te passen. Dit wordt gedaan doormiddel een systeem van Nodes en Transitions tussen verschillende Nodes. Hierdoor is op een gemakkelijke manier een decision tree weer te geven.

Een Node Editor kan gebruikt worden ter ondersteuning van een ontwikkeld systeem om regels, acties en beperkingen op te zetten.

5.4.4. Data file

Alle waardes die aanpasbaar moeten zijn kunnen naar een(of meer) bestanden geschreven worden. Deze waardes kunnen dan aan het begin van een spel opnieuw ingeladen worden en gebruikt worden om de bestaande objecten aan te passen naar de nieuwe situatie.

Hierbij is het erg gemakkelijk meerdere verschillende settings te gebruiken(doormiddel van aparte bestanden). Hierdoor is het gemakkelijk om bijvoorbeeld verschillende moeilijkheidsgraden te maken of verschillende thema's te gebruiken (bijvoorbeeld verschillende landen).

Dit systeem kan ook opgezet worden met een database, hierbij gaan wel extra kosten gepaard (server kosten).

5.5. Proof of concepts

Na de voorlopige technische analyse van de verschillende gevonden technieken zoals deze in de hier bovenstaande hoofdstukken te vinden zijn is er in overleg met de opdrachtgever besloten welke technieken verder uitgewerkt zullen worden.

Er is gekozen voor zowel MVC als ECS omdat beide Programming Paradigms ondersteuning bieden voor modularisatie en hergebruik van code. Dit ondersteunt de niet functionele eisen voor uitbreidbaarheid en onderhoudbaarheid van het project.

Er is gekozen voor MM om de mogelijkheden van Live aanpassingen aan Regels, Acties en Beperkingen te onderzoeken.

Er is gekozen voor FSM omdat dit een bestaande en veelgebruikte oplossing is voor vergelijkbare systemen.

Verdere informatie over de eisen en wensen aan het project zijn te vinden in [Bijlage VI: User Requirement Specification](#).

Verder is er bij alle technieken en methodes rekening mee gehouden dat deze te implementeren moeten zijn in het bestaande project. Het is niet mogelijk (of gewenst) het volledige project opnieuw te ontwikkelen.

De proof of concept(s) worden beoordeeld op de mogelijkheden die zij bieden om het bestaande bedrijfstypes na te bouwen. Er is gekozen om met het testen van de technieken en methodes uit te gaan van het bestaande bedrijfsgedrag omdat in de toekomst bedrijven meer gedrag kunnen krijgen maar dit gedrag niet complexer wordt als dat het nu is.

Waar mogelijk wordt geprobeerd bestaande proof of concept(s) te gebruiken als validatie van de gevonden technieken en methodes.

5.5.1 Entity Component System

Het proof of concept met betrekking tot ECS ontwikkelt tijdens het onderzoeksproject.

Het doel van deze proof of concept is het onderzoeken van de mogelijkheden die ECS biedt voor het opzetten van dynamisch bedrijfsgedrag. Hierbij wordt er gekeken naar de herbruikbaarheid van component en de hoeveelheid werk die iedere component is.

Om dit te bereiken zullen de bestaande acties voor het inkopen, verkopen en converteren van grondstoffen nagebouwd worden.

Voor deze proof of concept wordt gebruik gemaakt van een fictieve markt die een standaard prijs voor inkoop en verkoop hanteert. Verder wordt er gebruik gemaakt van de Trustee klasse van het bestaande spel. Deze dient vooral als ondersteuning van het te ontwikkelen proof of concept om het later (mocht dit noodzakelijk zijn) gemakkelijker te maken dit systeem in het bestaande spel te integreren.

Dit proof of concept is ontwikkeld aan de hand van het klasse diagram zoals het te vinden is in de Bijlage V: Klasse diagram POC ECS. Om de Acties voor Inkoop, Verkoop en Converteren van grondstoffen te kunnen realiseren zijn de components zoals die in Tabel 2 staan gemaakt.

Component	Beschrijving
BuyResource	Koop een x aantal van een bepaalde grondstof van de Market.
SellResource	Verkoop een x aantal van een bepaalde grondstof aan de Market.
ConvertResource	Converteert een x aantal van een (of meerdere) grondstoffen naar een andere grondstof.
CompanyController	Houd een lijst van Components bij en voert deze achter elkaar uit.
Storage	Slaat verschillende grondstoffen (tijdelijk) op.
CheckMoney	Controleert of het geld van een Trustee boven x is.
CheckDemand	Controleert of er een vraag is naar een specifieke grondstof (dit is momenteel altijd waar).

Tabel 2: Components

Het uitbreiden van dit prototype is gemakkelijk doordat het mogelijk is nieuwe components toe te voegen of bestaande components simpelweg te vervangen. Hierdoor voldoet het goed aan de gestelde niet functionele eisen.

De Check Components zouden gecombineerd kunnen worden tot een algemene component “ConditionCheckComponent” die gebruik maakt van dynamische condities. Hierdoor is er minder code duplication in het project en blijft het project overzichtelijker.

Het ontwikkelen van components gaat relatief snel doordat components verantwoordelijk zijn voor 1 actie (“single responsibility”).

5.5.2. Finite State Machine

Het proof of concept met betrekking tot FSMs is ontwikkeld tijdens het onderzoeksproject.

Het doel van deze proof of concept is het onderzoeken van de mogelijkheden die FSMs bieden voor het opzetten van dynamisch bedrijfsgedrag. Om dit te bereiken zullen de bestaande acties voor het inkopen, verkopen en converteren van grondstoffen nagebouwd worden.

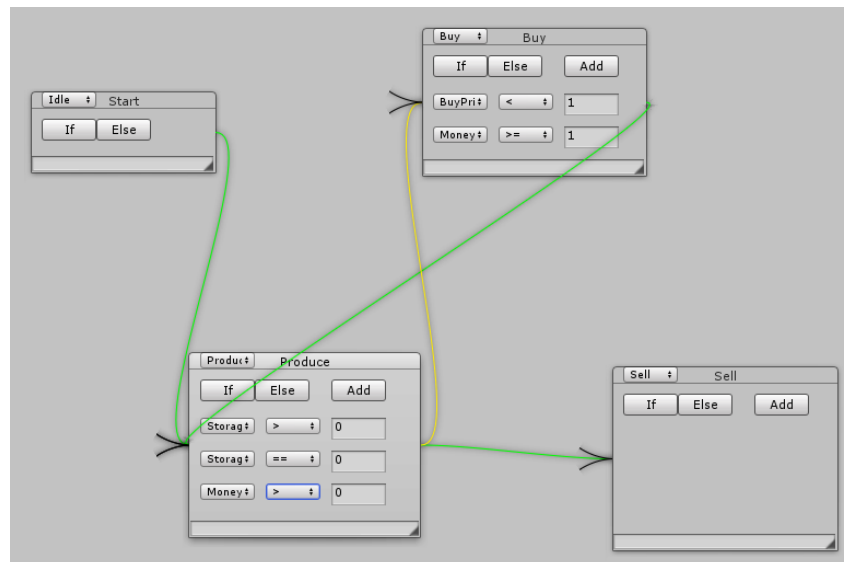
Voor deze proof of concept wordt gebruik gemaakt van een fictieve markt die een standaard prijs voor inkoop en verkoop hanteert. Deze FSM is ontwikkeld aan de hand van het klasse diagram zoals het te vinden is in de [Bijlage IV: Klasse diagram POC FSM](#).

Het proof of concept bevat een state machine met states die in Tabel 3 staan.

State	Beschrijving
Buy	Koop een x aantal van een bepaalde grondstof van de Market.
Convert	Converteert een x aantal van een (of meerdere) grondstoffen naar een andere grondstof.
Sell	Verkoop een x aantal van een bepaalde grondstof aan de Market.
Idle	Standaard wacht state waarin de FSM inactief is.

Tabel 3: States

Als interface voor de FSM is een Node Editor ontwikkeld als interface, een voorbeeld van het ontwikkeld systeem is te vinden in Figuur 5. Hiervoor is in overleg met de opdrachtgever gekozen omdat een Node Editor overzichtelijk veel verschillende states en de transities hiertussen kan weergeven.



Figuur 5: Voorbeeld Node Editor

Iedere Node in de Editor staat voor 1 specifieke state, hierdoor is in een overzicht te zien welke transitie tussen states bestaan en aan welke condities deze transitie moeten voldoen. Dit maakt het aanpassen van gedrag wat bij een bedrijf hoort gemakkelijker.

Iedere state staat voor 1 actie die het bedrijf kan uitvoeren, voordat het bedrijf de actie kan uitvoeren moet deze eerst voldoen aan alle gestelde Condities. Iedere state heeft 2 transitie een voor als de actie succesvol uitgevoerd is en een voor als de actie niet uitgevoerd kan worden. Als een Node voldoet aan al zijn Condities wordt de "if" transitie gebruikt(nadat de actie uitgevoerd is) zo niet dan wordt de "else" transitie gebruikt.

Het proof of concept is geschikt voor het maken van een systeem van regels, acties en beperkingen met betrekking tot de bedrijven omdat ieder bedrijfstype zijn eigen State Machine kan gebruiken.

Als nadeel heeft dit systeem dat het uitbreiden van dit proof of concept alleen mogelijk is door het programmeren van nieuwe states. Met een goede opzet van de interface en onderliggende klassen kan dit nadeel zoveel mogelijke beperkt worden door nieuwe states automatisch weer te geven en te kunnen gebruiken.

5.5.3. Micro Machinations

Adapt Tower is ontwikkeld door Ludomotion als ondersteuning van het MM onderzoek. Voor de ontwikkeling van Adapt Tower is gebruik gemaakt van XNA en de Phantom Game engine. De Phantom game engine is eveneens ontwikkeld door Ludomotion en maakt gebruik van het ECS paradigm. Momenteel is Adapt Tower het enige project dat gebruik maakt van MM.

In Adapt Tower is het voor de speler de bedoeling zich te verdedigen tegen random gespawnde vijanden. Dit kan gedaan worden door het bouwen van turrets en/of bases, het bouwen van deze voorwerpen kost de speler geld. De turrets schieten op vijanden, bases verzamelen geld, vijanden laten geld vallen als zijn vernietigd worden.

Adapt Tower maakt gebruik van MM diagrammen voor het modelleren van de in game Entiteiten(Turret, Base, Vijand). Adapt Tower geeft de speler de mogelijkheid tussen verschillende MM diagrammen te kiezen en hierdoor het gedrag van de entiteiten live aan te passen.

Dit proof of concept wordt bekeken om de mogelijkheden van MM met betrekking tot het live aanpassingen van game entiteiten te beoordelen. Om dit te bereiken wordt er gekeken naar de

opbouw van entiteiten binnen Adapt Tower. Verder wordt er gekeken naar hoe het inladen en gebruiken van modificatie bestanden werkt.

De entiteiten binnen Adapt Tower zijn gemaakt aan de hand van de klasse diagram zoals deze te vinden is in de [Bijlage IIV: Klasse diagram POC Adapt Tower](#).

Door het gebruik van ECS binnen dit proof of concept is het gemakkelijk uit te breiden met nieuwe functionaliteit en entiteiten, ook sluit het goed aan op de gestelde niet functionele eisen van het project. Ook sluit dit proof of concept goed aan bij de doelstellingen van het project door de mogelijkheid van MM om diagrammen gemakkelijk te testen doormiddel van simulaties(zowel versnelt spelen als auto run).

MM voor het inladen van nieuwe diagrammen uit doormiddel van een complexe DLL en hier is verder geen onderzoek naar gedaan. Na het inladen van een nieuwe diagram wordt de benodigde aanpassingen verstuurd naar de betreffende entiteiten doormiddel van Events.

Het nadeel van MM is dat deze momenteel nog in ontwikkeling is waardoor er geen Unity3D compatible versie beschikbaar is, hierdoor kan de ontwikkeling van het prototype eventueel vertraging oplopen.

5.5.4. Model View Controller

Dit proof of concept is ontwikkeld door Paul Brinkkemper. Het proof of concept heeft als doel een spel te realiseren waarin de gebruiker uitgelegd wordt hoe omgevingswetgeving werkt en wat het doel hiervan is. Binnen het spel bestaan een aantal entiteiten die ieder zijn of haar voorkeuren heeft met betrekking tot hoe de publieke ruimte ingedeeld/gebruikt wordt. Hierbij is het doel van de speler om de ruimte zo in te delen dat zo veel mogelijk mensen het hiermee eens zijn. Om dit te bereiken stemt iedere entiteit op voorgestelde indelingen en kunnen entiteiten deze ook zelf voorstellen, dit heeft als doel de ruimte met zoveel mogelijk samenspraak in te delen.

Het doel van dit proof of concept is om het gebruik van MVC en Data files te beoordelen. Hierbij wordt gekeken naar de mogelijkheden die deze technieken bieden om waarden van entiteiten in te laden. Hierbij wordt voornamelijk gekeken naar de mogelijkheden om nieuwe waarden toe te voegen.

Het proof of concept laadt alle entiteiten (Mensen, Wensen, Plaatsen, Gebouwen) in doormiddel van een data file aan het begin van het spel. Dit maakt het mogelijk om nieuwe instanties van entiteiten in te voegen of bestaande instanties waar nodig aan te passen. Dit systeem maakt het niet mogelijk nieuwe entiteiten toe te voegen aan het spel.

Het toevoegen van nieuwe variabelen waarop de entiteiten reageren wordt door het gebruik van een data file vergemakkelijkt, ook kan dit (in het geval van Excel of CSV) overzichtelijk weergegeven worden in een format waarmee de designers gemakkelijk en snel aanpassingen kunnen realiseren.

5.6. Gekozen oplossing

Aan de hand van de bovenstaande proof of concepts en in overleg met de opdrachtgever zijn naar de voor- en nadelen van alle mogelijk technieken gekeken. De proof of concept(s) worden beoordeeld op het de hoeveelheid werk en tijd die het kost om Regels, Acties en Beperkingen aan te passen (of toe te voegen).

Als beoordeling van de proof of concept(s) is een inschatting gemaakt van hoelang (in tijd) de hier onderstaande aanpassingen zouden kosten met de betreffende techniek en/of methode. De waarden hiervan zijn te vinden in Tabel 4.

Aanpassen Actie

Het aanpassen van de Converteer Actie (te vinden in [Bijlage VI: URS](#)) om per geproduceerde grondstof een bepaald percentage kans te hebben dat de conversie mislukt. Hierdoor raak het bedrijf zowel de input als de output grondstof kwijt.

Nieuwe Actie

Creëren van een nieuwe Actie die het bedrijven toestaat om grondstoffen op te slaan totdat zij ervoor een x percentage winst maken. Bedrag hiervoor wordt gebaseerd op de totale inkoop prijs + personeelskosten.

Aanpassen Regel

Het aanpassen van de CanDoLoanPayment Regel (te vinden in [Bijlage VI: URS](#)) om rekening te houden met de verwachte verhoging van inkomsten die een upgrade met zich meebrengt

Nieuwe Regel

Creëren van een nieuwe Regel die ervoor zorgt dat bedrijven alleen een upgrade kunnen beginnen (lening afsluiten) al alle geproduceerde grondstoffen verkocht worden.

Nieuwe Grondstof

Toevoegen van een nieuwe grondstof.

Nieuw Bedrijf

Een nieuwe type bedrijf toevoegen die gebruik maakt van een bestaande grondstof als input en een nieuw gecreëerde grondstof als output.

	Bestaande systeem	Entity Component Systeem	FSM	MM	MVC
Aanpassen Actie	+/- 2 uur	+/- 10 minuten	+/- 10 minuten	+/- 10 minuten	+/- 1 uur
Nieuwe Actie	+/- 4 uur	+/- 1 uur	+/- 2 uur	+/- 1 uur	+/- 3 uur
Aanpassen Regel	+/- 2 uur	+/- 5 minuten	+/- 10 minuten	+/- 10 minuten	+/- 5 minuten
Nieuwe Regel	+/- 4 uur	+/- 30 minuten	+/- 1 uur	+/- 30 minuten	+/- 2 uur
Nieuwe Grondstof	+/- 1 dag	+/- 5 minuten	+/- 30 minuten	+/- 5 minuten	+/- 5 minuten
Nieuw Bedrijf	+/- 1 dag	+/- 10 minuten	+/- 10 minuten	+/- 5 minuten	+/- 5 minuten

Tabel 4: Tijd per proof of concept

Voor het ECS en FSM zijn deze waarden bepaald door het doorvoeren van de genoemde aanpassingen. Voor MM en MVC is dit gebaseerd op een inschatting aan de hand van de architectuur.

Aan de hand van de bovenstaande analyse en in overleg met de opdrachtgever is er gekozen om gebruik te gaan maken van ECS. Er is voor ECS gekozen omdat deze techniek het toestaat om Regels, Acties en Beperkingen zo op te bouwen dat deze gemakkelijk te onderhouden en uitbreiden zijn. Ook is hiermee het hergebruiken en toevoegen van components gemakkelijk mogelijk.

Het creëren van nieuwe components zal nog steeds door een programmeur gedaan moeten worden. Maar dit systeem staat het toe om doormiddel van een interface nieuw bedrijfsgedrag te creëren door het rangschikken van Regels, Acties en Beperkingen.

Naast het genoemde Entity Component Systeem zal er gebruik gemaakt worden van MM. MM wordt gebruikt om aanpassingen van Regels, Acties en Beperkingen live te kunnen doen. Dit staat het toe om het balanceren en uittesten van MMD te versimpelen.

De gevonden technieken en methodes voor het visuele aspect van het project voldoen allemaal aan de gestelde eisen en wensen van het project. De keus voor de te gebruiken technieken en methodes wordt per iteratie genomen om zo de best mogelijke oplossing te kunnen realiseren voor de betreffende te ontwikkelen functionaliteit. Deze keus wordt genomen aan de hand van de eisen en wensen van de opdrachtgever aan de iteratie. Hierdoor kan het voorkomen dat er verschillende oplossingen gebruikt worden voor verschillende aspecten van het project.

6. Resultaten

Om aan de doelstellingen van het project te kunnen voldoen moeten de systemen die in Tabel 5 staan ontwikkeld worden. De systemen kunnen waar nodig in verschillende iteraties ontwikkeld worden. De stappen die ondernomen worden tijdens de iteratie(s) zijn beschreven in het Hoofdstuk 3.5.

Systemen	
Grondstofoverzicht	Grondstoffen toevoegen, nieuwe grondstoffen toevoegen aan gegenereerde markt grafieken.
Bedrijfsoverzicht	Bedrijven toevoegen, bestaande benodigde waardes van bedrijven aanpassen.
Dynamisch gedrag	Gedrag van een bedrijfstype aanpassen doormiddel van een interface.
Micro Machinations	Micro Machinations implementeren binnen Unity.

Tabel 5: Iteraties

In overleg met de opdrachtgever is bepaald in welke volgorde de systemen ontwikkeld worden (Dit is de volgorde waarop ze in de tabel staan). Er is voor gekozen eerst de systemen zonder MM te ontwikkelen. Hiervoor is gekozen omdat MM momenteel niet te integreren is in Unity3D. Ook is het niet zeker of dit in de toekomst (nadat het project afgerond is) wel mogelijk is. In overleg met de ontwikkelaar van MM wordt er gekeken naar de mogelijkheden om MM te integreren in Unity3D. Mocht deze integratie van MM niet succesvol zijn dan kunnen de eerder ontwikkelde systemen alsnog functioneren en hiermee de doelstellingen van het project behalen.

In dit hoofdstuk worden de stappen beschreven die ondernomen zijn om de betreffende systemen te ontwikkelen.

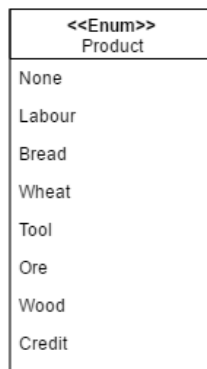
6.1. Grondstofoverzicht

Het eerste systeem is gericht op het aanpasbaar maken van de grondstoffen die binnen het spel gebruikt worden. Hierbij is het doel om nieuwe grondstoffen toe te kunnen voegen en bestaande grondstoffen aan te kunnen passen (of wanneer nodig verwijderen). Aanpassen van grondstoffen hoeft niet live mogelijk te zijn, wel moet dit met simpele handelingen binnen de Unity Editor (runtime) mogelijk zijn.

Er is voor gekozen om met dit systeem te beginnen omdat het aanpassen of toevoegen van een grondstof een grote impact heeft op de rest van het systeem. Dit systeem is te realiseren met betrekkelijk weinig handelingen waardoor het geschikt is om in een iteratie te ontwikkelen.

6.1.1. Huidige systeem

Grondstoffen worden opgeslagen in de in Figuur 6 afgebeelde Enum. Dit zijn alle grondstoffen die geproduceerd en verhandelt kunnen worden.



Figuur 6: Product Enum

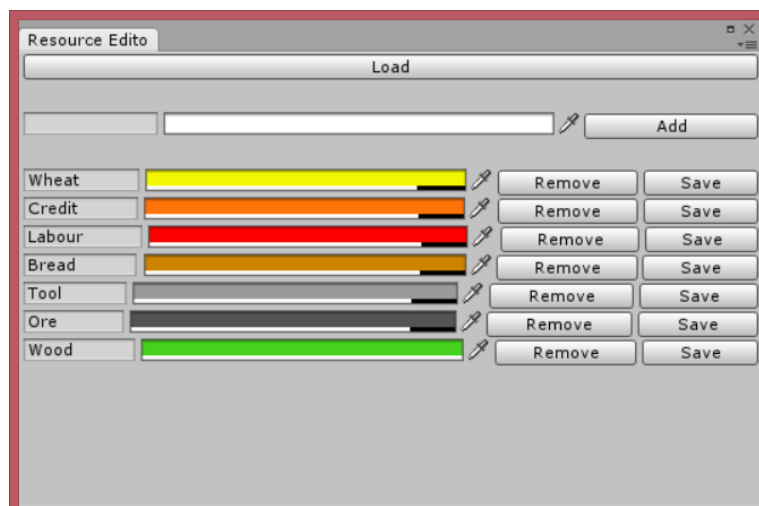
Hierop zijn de volgende 2 uitzonderingen: “None” wordt gebruikt voor bedrijven die geen grondstoffen produceren (Bijvoorbeeld Banken) en “Credit” wordt gebruikt als placeholder om de hoeveelheid actief krediet in de markt visualisatie weer te kunnen geven.

6.1.2. Nieuwe systeem

Voor het dynamisch toevoegen en gebruiken van grondstoffen heb ik ervoor gekozen om deze in een datafile op te slaan. Ik heb voor deze oplossing gekozen omdat deze oplossing runtime werkt in tegenstelling tot bijvoorbeeld een dynamisch aangemaakte enum. Hiervoor wordt gebruik gemaakt van StreamWriter en StreamReader om de gemaakte grondstoffen te kunnen opslaan en inladen in een Dictionary.

6.1.3. UI

Voor de interface heb ik gekozen om een Editor Window te gebruiken voor het weergeven en aanpassen van grondstoffen zoals het te zien is in Figuur 7. Deze interface kan hierdoor alleen in de Editor gebruikt worden waardoor het live aanpassen van grondstoffen momenteel niet mogelijk is.



Figuur 7: interface toevoegen grondstof

De grondstof “None” is niet opgenomen in deze interface omdat deze momenteel moet bestaan om Bank entiteiten binnen het spel mogelijk te maken. Hierdoor kan deze niet aangepast/verwijderd worden.

6.1.4. Markt visualisatie

In game worden een aantal waarden weergegeven doormiddel van grafieken het gaat hierbij voornamelijk over de waarde en beschikbare hoeveelheid van de grondstoffen. Een voorbeeld hiervan is te vinden in Figuur 8.



Figuur 8: Markt visualisatie

Het dynamisch aanmaken van grondstoffen maakt het noodzakelijk dat de marktvisualisatie ook aangepast wordt om de waarden van de nieuwe grondstoffen dynamisch weer te geven.

Om dit te bereiken maakt de Market de benodigde PriceDataStorage objecten en de benodigde objecten (GraphLine, GraphButton) voor tekenen van de grafieken aan op het moment dat deze opgestart wordt. Hierdoor is het mogelijk om nieuw toegevoegde grondstoffen direct te zien in de grafieken.

De kleur van de grafiek lijn kan ingesteld worden bij het aanmaken/aanpassen van de grondstoffen.

6.2. Bedrijfsoverzicht

Het tweede systeem is erop gericht de bestaande entiteiten in MMD aanpasbaar te maken. Dit systeem dient als voorbereiding op het aanpassen van het bedrijfsgedrag. Daarom moet het systeem gemakkelijk aanpasbaar zijn of waar mogelijk niet aangepast hoeft te worden om het nieuwe systeem voor bedrijfsgedrag te ondersteunen.

In het huidige systeem zijn aanpassingen aan entiteiten mogelijk doormiddel van de Unity Inspector. Deze aanpassingen kunnen gerealiseerd worden per bedrijfstype (prefab).

Het nieuwe systeem maakt gebruik van 2 Editor Windows om aanpassingen mogelijk te maken. Wel blijven aanpassingen alleen mogelijk per bedrijfstype dit is in overleg met de opdrachtgever besloten. Hiervoor is gekozen omdat in zowel de digitale als fysieke versie van Money Maker het bedrijfstype leidend is voor de acties die een bedrijf kan ondernemen. Mocht het nodig zijn dat instanties van het zelfde bedrijfstype verschillend gedrag moeten vertonen kan dit opgelost worden door meerdere bedrijfstypes te realiseren met het zelfde uiterlijk maar andere Regels, Acties en Beperkingen.

Dit systeem wordt ontwikkeld in 2 iteraties (1 per Editor Window).

6.2.1. Company Overview

Het Company Overview Window zoals het te zien is in Figuur 9 laat een overzicht zien van alle bestaande bedrijfstypes. Voor ieder bedrijf in dit overzicht worden de in- en output grondstoffen en de afbeelding die bij het bedrijf hoort weergegeven. Deze waarden geven een duidelijk beeld van de bedrijven en hebben het meeste effect binnen het spel (balancing).

De bedrijven worden opgehaald vanuit een gespecificeerde folder. Hierdoor is het mogelijk om gebouwen zowel via de geleverde interface als Unity toe te voegen.

The 'Company Edito' window displays a list of companies and their associated needs and offers. At the top, there are 'Load' and 'Add' buttons. Below this, a 'Numbers of Company's: 4' label is shown. The 'Resource Count: 8' is also indicated. The companies are listed on the left with their respective icons: Farm (yellow house), House (red house), Mill (blue mill), and Woodcutter (green house). Each company has a 'Needs' and an 'Offers' section, both with 'Product / Amount' labels. The 'Needs' section for each company shows the products they require, and the 'Offers' section shows the products they provide. Each entry has a spin button and a text field for the amount.

Company	Needs Product / Amount	Offers Product / Amount
Farm	Wheat: 1	Labour: 3
House	Credit: 1, Ore: 1	Wheat: 2
Mill	Wheat: 1, Labour: 3	Credit: 6
Woodcutter	Wheat: 5	Ore: 6

Figuur 9: Company Overview

Dit overzicht maakt het mogelijk nieuwe bedrijfstypes toe te voegen en bestaande bedrijfstypes te verwijderen. Bij nieuwe bedrijven wordt automatisch een Company en Trustee Component toegevoegd.

Door de afbeelding van een bedrijfstype te selecteren wordt het Company Detail View Window geopend voor dit specifieke bedrijf.

6.2.2. Company Detail view

Het Company Detail View Window zoals het te zien is in Figuur 10 laat een gedetailleerd overzicht zien van het geselecteerde bedrijf. In dit overzicht worden alle publieke variabelen en de hiervan waarden weergegeven, deze waarden worden gesorteerd op Component en Type. Er is voor gekozen de waarden te sorteren op Components omdat dit ervoor zorgt dat er geen verwarring kan ontstaan als meerdere components dezelfde variabele namen gebruiken.

The 'CompanyViewF' window displays detailed information for a selected company. At the top, there is a 'House' label and an 'Image:' field with a 'Select' button. Below this, there are 'Needs' and 'Offers' sections, both with 'Product / Amount' labels. The 'Needs' section shows the products the company requires, and the 'Offers' section shows the products the company provides. Each entry has a spin button and a text field for the amount. There are also 'X' and 'S' buttons next to each entry. Below the 'Needs' and 'Offers' sections, there is a 'Save' button. The 'Fields' section contains various variables and their values, sorted by Component and Type. The variables are: string (trusteeName: Farm1, companyName: , entrepreneurType:), int (Product: 1, currentUpgradeTime: 0, UpgradeTime: 2, UpgradeLevel: 1, bustTime: 0, priceCalculationDay: 0, purchaseCalculationDay: 0), and float (UpgradeMultiplier: 1.5).

Doormiddel van Reflection worden alle publieke variabelen en de waardes hiervan opgehaald. Dit wordt gedaan voor alle scripts die aanwezig zijn op de bedrijfsprefab.

Aanpassingen aan de weergegeven Fields worden doorgevoerd doormiddel van de “Save” knop. Zodat het systeem deze niet iedere frame probeert te updaten, hiervoor is gekozen in verband met de performance. Aanpassingen aan publieke variabelen worden ook gedaan doormiddel van Reflection.

6.3. Dynamisch gedrag

Het derde systeem is erop gericht om de bestaande architectuur die verantwoordelijk is voor de Regels, Acties en Beperkingen van entiteiten te vervangen door een systeem dat gebruik maakt van ECS.

Het gewenste systeem wordt gerealiseerd in meerdere iteraties, het ontwerp voor het systeem is gemaakt in de eerste iteratie en waar nodig aangepast. Ook worden de iteraties benut om waar nodig en mogelijk het bestaande project en code op te schonen. Dit kan bereikt worden door het herstructureren van code/assets en het verwijderen van ongebruikte code/assets.

6.3.1. Ontwerp

Het te ontwikkelen systeem wordt opgedeeld in Components en Actions. Components geven een entiteit de mogelijkheid om stellen entiteiten in staat bepaald gedrag te ondersteunen, Actions voeren het betreffende gedrag uit.

De Components en Actions zijn ontwikkeld aan de hand van het klasse diagram zoals het te vinden is in [Bijlage III: Klasse diagram systeem](#). Components en Actions worden beiden in hun eigen iteratie ontwikkeld.

De Markt zoals die momenteel bestaat wordt niet omgezet naar Components, waar mogelijk wordt deze klasse wel opgeschoond en uitgebreid.

De Trustee klasse wordt waar mogelijk omgezet naar Components die zelfstandig gebruikt kunnen worden. Tijdens de ontwikkeling van het Component systeem wordt wel gebruik gemaakt van deze klasse om de uiteindelijke integratie van het systeem soepel te laten verlopen.

6.3.2. Components

De Components worden ontwikkeld als vervanging van de bestaande functionaliteit van bedrijven. Iedere Component die ontwikkeld is wordt direct geïntegreerd in het bestaande systeem. Hierdoor kan functionaliteit verwijderd worden uit de Trustee en Company klassen.

Tijdens deze iteratie zijn de Components die in Tabel 6 staan ontwikkeld.

Components	Omschrijving
CompanyController	Controller die kan worden aangeroepen door de GameManager. Houd een lijst bij van alle actions die een bedrijf kan ondernemen in de juiste volgorde.
Upgrader	Houd een lijst bij mogelijke upgrades voor een bedrijf.
Storage	Maakt het mogelijk een of meerdere grondstoffen op te slaan. Hoeveelheid grondstoffen en limiet hiervan is aanpasbaar.

Tabel 6: Components

Er zijn geen components ontwikkeld als ondersteuning van Markten, Banken en andere Entiteiten binnen MMD.

6.3.3. Actions

De Actions worden ontwikkeld als uitbreiding op de eerder ontwikkelde Components. De Actions voeren het betreffende bedrijfsgedrag uit. Hiervoor maken ze gebruik van de functionaliteit die de Components aanbieden.

Tijdens deze iteratie zijn de Actions die in Tabel 7 staan ontwikkeld.

Action	Omschrijving
BuyFromMarket	Koop grondstoffen van de markt.
SellToMarket	Verkoop grondstoffen aan de markt.
ConvertResource	Converteert x hoeveelheid van een (of meer) grondstoffen naar een (of meer) andere grondstoffen.
Upgrade	Gegevens voor 1 specifieke upgrade. Effect op in- en output grondstoffen. Kosten in grondstoffen.
CheckUpgrade	Kijkt aan de hand van conditions of een upgrade uitgevoerd kan/moet worden.

Tabel 7: Actions

Als ondersteuning van de Acties is een EntityManager klasse ontwikkeld. De EntityManager stelt Actions en Components in staat om entiteiten op te zoeken doormiddel van het type, naam of ID.

Er is nog geen ondersteuning gebouwd voor het verstrekken/aannemen van leningen dor bedrijven.

6.4. Micro Machinations

Dit systeem heeft als doel het integreren en implementeren van MM in MMD. MM wordt gebruikt om aanpassingen aan bedrijven Live te kunnen doorvoeren. Dit gekoppeld aan de mogelijkheid om bedrijven te simuleren maakt het balanceren van bedrijven (en verschillende bedrijfsinstellingen) gemakkelijker.

MM wordt momenteel ontwikkeld door Riemer van Roozen als onderdeel van het “Live Intelligent Visual Environments for Game Design” onderzoek. MM is gemaakt in C++.

6.4.1. Messaging

MM maakt gebruik van Events om aanpassingen aan diagrammen door te voeren in de entiteiten. Hierom wordt er voor het Unity project een MessageHandler en Message klasse ontwikkeld.

MM verstuurd Events voor het Toevoegen, Verwijderen en Updaten van objecten in een diagram. Verder worden er ook Events verstuurd voor het aanpassen van variabelen in een diagram.

De Message klasse wordt gebruikt om gegevens van deze Events op te slaan totdat deze gebruikt worden. De MessageHandler klasse wordt gebruikt om Events te ontvangen en versturen tussen Entiteiten. Verder is deze ook verantwoordelijk voor het doorsturen van de messages naar de juist entiteit dit kan gedaan worden aan de hand van het entiteit ID.

6.4.2. Unity Integratie

Het integreren van MM in Unity is met het aangeleverde project niet mogelijk omdat dit een Unmanaged C++ project is. Hierdoor is het niet mogelijk om de bestaande code in Unity te importeren en aan te roepen.

Om alsnog gebruik te kunnen maken van MM in Unity wordt hiervoor een C++ / Common Language Infrastructure (CLI) wrapper ontwikkeld. CLI maakt het mogelijk om een managed laag boven op een unmanaged dll (project) te creëren waardoor deze wel te gebruiken is binnen Unity (of andere .Net projecten).

7. Conclusies

De uitvoer van het project heeft inzicht geboden in de verschillende technieken en methodes om een dynamisch systeem van Regels, Acties en Beperkingen te realiseren. Hierdoor kan de gestelde onderzoeksvraag beantwoordt worden.

Hoe kan een systeem ontwikkeld worden dat het mogelijk maakt om regels, acties en beperkingen van entiteiten binnen het spel MoneyMaker Deluxe aan te passen door een designer?

In het bestaande systeem van Regels, Acties en Beperkingen maken de entiteiten gebruik van de acties: Inkopen, Converteren en Verkopen. Aan deze acties zitten condities verbonden waaraan voldaan moet worden voordat een bedrijf deze actie kan uitvoeren. Het niet kunnen uitvoeren van een actie zorgt ervoor dat het gehele proces niet uitgevoerd kan worden. De condities die verbonden zijn aan de Acties (en de acties zelf) zijn voor alle entiteiten het zelfde, de waardes voor deze condities kunnen per entiteit veranderen.

Het nieuwe systeem moet Firebrush Studios in staat stellen efficiënt en overzichtelijk aanpassingen te realiseren aan de Acties, Regels en Beperkingen die gebruikt worden door de entiteiten. Hiervoor zijn een aantal bestaande technieken en methodes beschikbaar. Het gaat hier om technieken om AI systemen te realiseren en die te gebruiken om gedrag te creëren of technieken die gebruikmaken van modularisatie om gedrag op te delen in kleine losstaande onderdelen die onderhoudbaar, uitbreidbaar en herbruikbaar zijn.

Er is gekozen om voor het nieuwe systeem gebruik te maken van ECS dit is een programming paradigm. ECS deelt de Regels, Acties en Beperkingen op in “single responsibility” components. Dit heeft als voordeel dat er weinig (of geen) dependencies zijn tussen verschillende components, dit heeft als voordeel dat de components gemakkelijk te onderhouden en aan te passen zijn. Deze opbouw maakt het mogelijk gemakkelijk nieuwe Regels, Acties en Beperkingen te creëren doordat components verantwoordelijk zijn voor een actie (gefocust en afgebakend).

Naast ECS wordt er gebruik gemaakt van MM om aanpassingen aan Regels, Acties en Beperkingen live mogelijk te maken.

Om het nieuwe systeem te kunnen realiseren moet een groot gedeelte van het bestaande project aangepast worden het gaat hierbij de code die verantwoordelijk is voor het bestaande economische systeem.

Dit wordt vervangen door het nieuwe systeem van Regels, Acties en Beperkingen gemaakt doormiddel van ECS. Door het samenvoegen van een reeks van components kan entiteit gedrag dynamisch gecreëerd worden.

Het nieuwe systeem gekoppeld aan complete en duidelijk documentatie moet ervoor zorgen dat in de toekomst het project gemakkelijker aanpasbaar en uitbreidbaar is.

Reflectie

Nu ik terugblik op het uitgevoerde onderzoekstraject is het me duidelijk geworden dat het project heel breed gedefinieerd was. Hierdoor miste er in de eerste weken van het project focus bij de uitvoering. Dit had voorkomen kunnen worden door in overleg met de opdrachtgever beter te definiëren wat het project inhoud en welke functionaliteit door mij ontwikkeld moest worden.

Verder heb ik tijdens de afstudeer periode noodgedwongen een aantal dagen (weken) thuisgewerkt doordat er geen computer aanwezig was waarop ik mijn werkzaamheden kon uitvoeren, dit was noodzakelijk door een defect aan mijn eigen laptop. Tijdens deze periode heb ik op kantoor geholpen met overige werkzaamheden verder heb ik waar mogelijk thuis gewerkt aan mijn onderzoek.

Door de vertragingen die het project opgelopen heeft ben ik genoodzaakt geweest om mijn tijd zo goed mogelijk in te delen en mij aan deze gestelde planning te houden. Dit waren tot nu toe voor mij altijd struikelblokken tijdens de afstudeer periode is dit heel erg verbeterd. Verder heb ik door het vele thuis werken geleerd gedisciplineerd aan gestelde taken te werken en te blijven werken.

Het andere struikelblok waarop ik mij tijdens deze stage gefocust heb is duidelijk overbrengen waarom iets wel of juist niet kan. Hierbij gebruik maken van duidelijk onderbouwde argumenten en duidelijk maken waarom het zo is.

Informatie over mijn professionele ontwikkeling tijdens de gehele afstudeer periode is te vinden in [Bijlage II: Tips & Tops](#).

Literatuurlijst

- Atwood, J. (2008, 05 05). *Understanding Model-View-Controller*. Opgehaald van codinghorror: <https://blog.codinghorror.com/understanding-model-view-controller/>
- Bevilacqua, F. (2013, 09 24). *Finite-State Machines: Theory and Implementation*. Opgehaald van gamedevelopment: <https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867>
- Dormans, J. (2012). *Engineering Emergence*.
- Dormans, J. (2016, 03 17). *Phantom*. Opgehaald van github: <https://github.com/ludomotion/phantom>
- Games, B. (2013, 04 02). *Understanding Component-Entity-Systems*. Opgehaald van gamedev: https://www.gamedev.net/resources/_/technical/game-programming/understanding-component-entity-systems-r3013
- Martin, A. (2007, 11 11). *Entity Systems are the future of MMOG Development*. Opgehaald van t-machine: <http://t-machine.org/index.php/2007/11/11/entity-systems-are-the-future-of-mmog-development-part-2/>
- Påhlsson, N. (2002). *An Introduction to Aspect-Oriented Programming and AspectJ*.
- Rozen, R. (2014, 12 05). *MM-Lib*. Opgehaald van github: <https://github.com/vrozen/MM-Lib>
- Rozen, R. (sd). *A Pattern-Based Game Mechanics Design Assistant*.
- Wenzel, J. (2011). *MVVM vs MVP vs MVC: The differences explained*. Opgehaald van joel.inpointform: <http://joel.inpointform.net/software-development/mvvm-vs-mvp-vs-mvc-the-differences-explained/>

Projectplan

Afstudeer stage Firebrush Studios

Utrecht



Datum	:	29-08-2016
Versie	:	0.5
Status	:	Status
Bestandsnaam	:	PID Firebrush
Auteur	:	Tom Vaessen

Versie

Versie	Datum	Auteur(s)	Wijzigingen	Status
0.1	16-09-2016	Tom Vaessen		
0.2	22-09-2016	Tom Vaessen		
0.3	30-09-2016	Tom Vaessen		
0.4	03-10-2016	Tom Vaessen		
0.5	13-10-2016	Tom Vaessen		

Verspreiding

Versie	Datum	Aan
0.1	19-09-2016	Mark van Kuijk
0.1	19-09-2016	Paul Brinkkemper
0.2	28-09-2016	Mark van Kuijk
0.3	02-10-2016	Paul Brinkkemper
0.3	02-10-2016	Mark van Kuijk
0.4	03-10-2016	Paul Brinkkemper
0.4	03-10-2016	Mark van Kuijk
0.5	14-10-2016	Paul Brinkkemper
0.5	14-10-2016	Mark van Kuijk

Inhoudsopgave

1.	Projectopdracht	42
1.1	Doel van het project	42
1.2	Doel van dit document	43
1.3	Begrenzing	43
1.4	Strategie	44
1.5	Onderzoeksplan	47
1.6	Randvoorwaarden	47
1.7	Productdecompositiestructuur	48
2.	Projectorganisatie	49
2.1	Teamleden	49
2.2	Communicatie	50
2.3	Besluitvorming	50
3.	Activiteiten en tijdplan	50
3.1	Opdeling en aanpak van het project	50
3.2	Overall tijdplan	51
3.3	Fase Project opzet	51
3.4	Fase onderzoek	52
3.5	Fase Iteratie(s)	53
3.6	Fase Afronding	54
4.	Kwaliteitsbewaking, Testen, Configuratiemanagement	55
4.1	Goedkeuringen	55
4.2	Testaanpak/strategie	55
4.3	Reviews	55
5.	Risico's en afhankelijkheden	56
5.1	Afhankelijkheden	56
5.2	Projecten die van dit project afhankelijk zijn	56
5.3	Risico's en uitwijkactiviteiten	56

Projectopdracht

Doel van het project

Firebrush Studios (de opdrachtgever) ontwikkelt MoneyMaker Deluxe, een economisch bank spel met als doelstelling spelers kennis bij te brengen van hoe het huidige banken systeem werkt en ontstaan is. Binnen het spel bestaan een aantal verschillende bedrijven (Bank, Boerderij, Markt, Bakker, Houthakker, Huis, Herberg). Deze bedrijven kunnen onderling een aantal acties uitvoeren. Verder kan de speler ook acties uitvoeren die invloed hebben op de bedrijven.

Entiteiten binnen MoneyMaker Deluxe werken doormiddel van een systeem van regels. Deze bepalen wanneer en onder welke voorwaarden de bedrijven acties mogen ondernemen. Hierbij gaat het over acties die een bedrijf binnen de economie kan doen (verkopen/inkopen van producten, tegen welke prijs, op welk moment).

Het huidige systeem van regels dat gebruikt wordt door de entiteiten is erg complex. Dit komt mede door de complexe code base en de grootte van het project. Hierdoor duren kleine aanpassingen aan bestaande regels of het toevoegen van nieuwe regels erg lang. Denk hierbij aan een paar dagen tot een paar weken afhankelijk van de te realiseren aanpassing. Aanpassingen aan het huidige systeem van regels moeten volledig in de code gerealiseerd worden. Hierdoor moeten deze aanpassingen altijd door een programmeur gedaan worden.

Door de lange tijd die nodig is voor het realiseren van deze aanpassingen ligt de iteratie tijd van het project hoog. Door de lange iteratie duur is het moeilijk aanpassingen aan regels of nieuwe regels te valideren. Hierdoor duurt het betrekkelijk lang voordat er bepaald kan worden of een regel wel of niet werkt.

Hierdoor is het lastig te controleren welke Educatieve doelen het spel bereikt. Ook is het hierdoor lastig snel aanpassingen te maken om meer Educatieve doelen te behalen.

Het systeem moet het mogelijk maken de volgende aanpassingen te maken binnen het spel:

- Aanpassen van bestaande bedrijven
 - Productie kosten
 - Productie aantal
- Maken van nieuwe bedrijven
- Maken van nieuwe grondstoffen
- Aanpassen bestaande handels regels
 - Wanneer kan een bedrijf inkopen/verkopen
 - Wat voor betaal middel kan een bedrijf gebruiken
 - Minimale inkoop/verkoop prijs
 - Maximale inkoop/verkoop prijs
- Toevoegen nieuwe regels
- Aanpassen spel regels

Een voorbeeld van een toevoeging die het systeem moet kunnen maken is als volgt. Er moet een nieuw bedrijf bijkomen die gaat dienen als een opslagloods. Andere bedrijven in het spel kunnen ervoor kiezen in dit bedrijf ruimte te huren om geproduceerde goederen op te slaan.

Hierdoor kan het bedrijf voor de grondstoffen op een later moment mogelijk meer geld voor krijgen (speculatie).

Doelstellingen

Het onderzoek wordt uitgevoerd als ondersteuning van de ontwikkeling van het spel. De ontwikkeling van het spel gaat verder nadat het onderzoek afgerond is. Hierna is er nog +- 1 jaar ontwikkeltijd waarin het spel afgemaakt moet worden.

De voornaamste doelstelling van het project is het verkorten van de iteratie duur voor de ontwikkeling van MoneyMaker Deluxe. Dit moet bereikt worden door het proces van aanpassingen aan de economie(en economische entiteiten) binnen het spel te versimpelen.

Het ontwikkelen van dit systeem versnelt de ontwikkel cyclus van het spel. Door aanpassingen aan de regels voor entiteiten te versimpelen kunnen er meer en vaker iteraties uitgevoerd worden. Door simpele aanpassingen aan de core mechanics van het spel mogelijk te maken, is het mogelijk om sneller de leerdoelen van het spel te realiseren.

Ook moet het systeem het mogelijk maken dat aanpassingen aan de regels gedaan kunnen worden door een designer. Om dit te kunnen realiseren moet er voor zover mogelijk een visuele interface gemaakt worden waarmee de eerder genoemde aanpassingen gedaan kunnen worden. Eisen aan de interface zijn op dit moment niet bekend.

Doel van dit document

Dit document is opgesteld met als doel de lezer te informeren van de scope van het project, ook dient dit document als basis voor de beoordeling van het project en gerealiseerde eindproduct(en).

Het document behandelt de volgende aspecten van het project:

- Wat moet het project bereiken
- Waarom moet dit bereikt worden
- Wie is er betrokken bij het project en wat zijn hun rollen en verantwoordelijkheden
- Wat zijn de afhankelijkheden en risico's van het project
- Wanneer moet het project gerealiseerd zijn

Begrenzing

De focus van het onderzoek/project ligt op het ontwikkelen van een designtool ter ondersteuning van de ontwikkeling van MoneyMaker Deluxe.

Het onderzoek wordt uitgevoerd binnen een Raak project. Eventuele eisen vanuit het raak project worden zo goed mogelijk in het onderzoek/project opgenomen. Hierbij is het wel van belang dat op eisen aan het onderzoek vanuit het opleidingsinstituut leidend zijn.

Firebrush Studios ontwikkeld een bordspel variant met de zelfde leerdoelen als MoneyMaker Deluxe. De ontwikkeling van het bordspel valt buiten de scope van mijn project.

Strategie

Er wordt binnen het bedrijf gebruik gemaakt van Trello voor het bijhouden van de planning. Binnen Trello worden de volgende lijsten gebruikt:

- Backlog - Een overzicht van alle userstories voor het project.
- Bugs - Een overzicht van alle huidige bekende bugs. Deze taken hebben de hoogste prioriteit.
- Sprint log - Een overzicht van userstories in de huidige spint.
- Doing - De taken waar momenteel aan gewerkt wordt.
- Done - Taken die afgerond zijn.
- Checked - Taken die getest zijn en klaar staan voor release .

Verder wordt er gebruik gemaakt van Toggle om bij te houden hoelang er aan taken besteed wordt. Voor het opslaan van documenten wordt gebruik gemaakt van Google Drive. De bestaande code base staat in een Git repository.

Iedere week is er een voortgangsgesprek met de opdrachtgever. Hierin wordt in grote lijnen de richting van het project in besproken. Hiernaast is er dagelijks informeel overleg over de specifieke taken waar aan gewerkt wordt. Eventuele andere gesprekken die nodig zijn om de opdracht of taken te definiëren wordt incidenteel gedaan wanneer noodzakelijk.

Voor de voorbereidende(PID, (voor)onderzoek) fases van het project wordt gebruik gemaakt van de waterval methode. Beide fases moeten afgerond zijn voordat er doorgestroomd kan worden naar de volgende fase.

Voor de onderzoeksfase wordt gebruik gemaakt van bieb-, veldonderzoek. Deze fase focus zich voornamelijk op het vaststellen van de state of the art en het zo breed mogelijk kijken naar mogelijke oplossingen

Het te ontwikkelen prototypen wordt gerealiseerd door een iteratief proces. Dit proces maakt gebruik van de volgende 5 fases:

Probleemstelling

In de deze fase wordt in overleg met de opdrachtgever bepaald wat de doelstelling is voor de komende iteratie. Ook wordt er bepaald welke technieken (zoals deze gevonden zijn tijdens het onderzoek) gebruikt gaan worden.

Bij enige onduidelijkheid over de te ontwikkelen functionaliteit worden deze in overleg met de opdrachtgever opnieuw gedefinieerd.

Diagnose

In deze fase ga ik mogelijke problemen en moeilijkheden formuleren aan de hand van doelstelling van de iteratie. Hiervoor zal gebruik gemaakt worden van veld-, lab- en showroom onderzoek.

Doormiddel van biedonderzoek ga ik op zoek naar best practices en design patterns voor het ontwikkelen van de gewenste features. Hierbij focust het onderzoek zich voornamelijk op het opstellen van een dynamische regels systeem binnen spellen.

Doormiddel van veld- en showroom onderzoek ga ik opzoek naar bestaande oplossingen en methodes om dit te realiseren. Hierbij wordt ook gekeken naar andere werkgebieden.

Het onderzoek binnen deze fase kan ook gebaseerd worden op gevonden resultaten van eerdere iteraties.

Plan

Deze fase wordt gebruikt om de te ontwikkelen features te ontwerpen. Hiervoor wordt gebruik gemaakt van interface schetsen en UML diagrammen. Afhankelijk van de te ontwikkelen features kunnen hiervoor nog andere ontwerpen nodig zijn.

Verder worden eventuele veranderingen aan de bestaande code base in deze fase gedocumenteerd.

Ook wordt er in deze fase mocht het nodig zijn een nieuwe branch van het project gemaakt. Dit om de originele code base of een al ontwikkeld prototype niet onnodig aan te passen.

Ingreep

In deze fase wordt het prototype ontwikkeld, of doorontwikkeld afhankelijk van de specifieke situatie. De ontwikkeling wordt gedaan aan de hand van het eerder gerealiseerde ontwerp.

Eventuele aanpassingen aan het gemaakte ontwerp of gewenste features wordt duidelijk gedocumenteerd.

Evaluatie

In deze fase wordt het ontwikkeld prototype getest en gevalideerd door de opdracht gever. Dit wordt gedaan om te beoordelen of het prototype voldoet aan de doelstellingen van de iteratie. Het gaat hier voornamelijk om de functionaliteit van het prototype en de bruikbaarheid van de interface.

Voor het valideren van het prototype is de opdrachtgever leidend, eventuele andere partijen kunnen wel de prototypes testen maar niet valideren. Voor het testen wordt gebruik gemaakt van veld- en labonderzoek in de vorm van user-testen. Tijdens deze testen zal de opdrachtgever proberen aanpassingen te maken aan het spel, de specifieke aanpassingen hangen af van de doelstellingen van de iteratie.

Eventueel gevonden tekortkomingen van het prototype kunnen als verbeterpunten meegenomen worden naar een volgende iteratie.

Onderzoeksplan

Onderzoeksvraag

Hoe kan een systeem ontwikkeld worden dat het mogelijk maakt om regels, acties en beperkingen van entiteiten binnen het spel MoneyMaker Deluxe doormiddel van een interface aan te passen?

Deelvragen

Op dit moment zijn de volgende deelvragen geformuleerd:

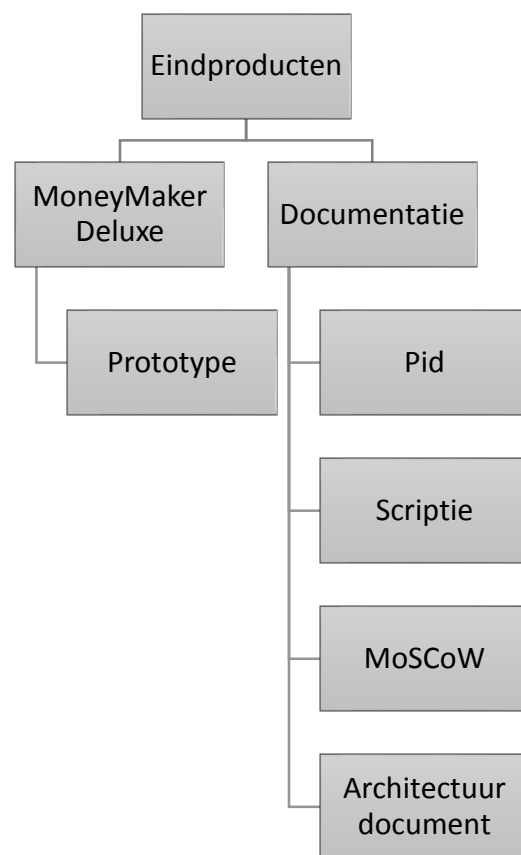
- Welke regels, acties en beperkingen moeten aanpasbaar zijn door het systeem?
- Welke technieken zijn er beschikbaar om een systeem van regels/acties/beperkingen te maken
- Welke technieken zijn er beschikbaar om dit realtime te kunnen doen?
- Welke technieken zijn er beschikbaar voor het visueel aanpassen van objecten?
- Hoe kunnen aanpassingen aan regels, acties en beperkingen gedaan worden zonder hiervoor te hoeven programmeren?
- Hoe kan het systeem zo opgebouwd worden dat het goed kan omgaan met aanpassingen aan de basisfunctionaliteit van het spel?
- Kan het systeem zo opgezet worden dat dit voor meerdere projecten te gebruiken is?

Bovenstaande deelvragen kunnen tijdens het onderzoek waar nodig nog gewijzigd worden.

Randvoorwaarden

Einddatum uiterlijk : 15-01-2017
Budget maximaal : N.v.t.
Overige voorwaarden : N.v.t.

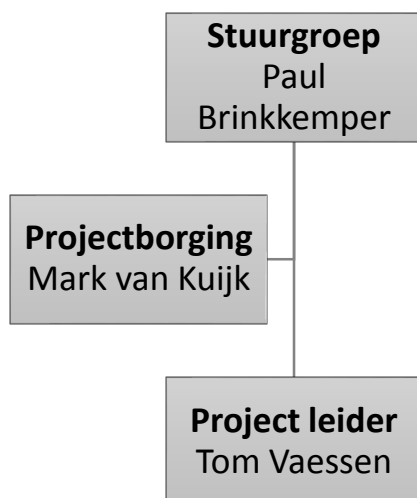
Productdecompositiestructuur



Figuur 1 Projectdecompositiestructuur

Projectorganisatie

Teamleden



Figuur 2 Projectorganisatie

Naam + tel + e-mail	Rol/taken
Tom Vaessen 0641090950 tom.vaessen.tv@gmail.com	Projectleider
Paul Brinkkemper 0612345678 paul@firebrushstudios.com	Bedrijfsbegeleider
Mark van Kuijk 0645967777 mark.vankuijk@fontys.nl	Docentbegeleider
Alexander Kappelhoff 0644093653 alex@firebrushstudios.com	Animater/Designer
Stefan Leijnen s.leijnen@hva.nl	Raak onderzoeker
Riemer van Rozen R.A.van.Rozen@cw.nl	Raak onderzoeker Technische ondersteuning MicroMachinations

Communicatie

Soort overleg	Frequentie	Aanwezig
Project voortgang	1 maal per dag	Projectleider
Stage voortgang	1 maal per 2 weken	Projectborging Projectleider
Overleg Raak project	Incidenteel	Projectleider Raak project leden

Hierboven worden communicatie momenten binnen het team weergegeven.

Besluitvorming

De projectleider is bevoegd om beslissingen te nemen over het gehele project hierbij wordt hij waar nodig ondersteund door de stuurgroep.

Wijzigingen aan het project of de beoogde eindproducten zullen in samenspraak tussen de projectleider en stagiaire afgehandeld worden, waarbij de eisen en wensen van de projectleider leidend zijn.

Indien er een conflict is tussen het bedrijf en stagiaire wordt er geprobeerd een compromis te vinden, hierbij kan ondersteuning gezocht worden van de docentbegeleider.

Activiteiten en tijdplan

Opdeling en aanpak van het project

Het project is opgedeeld in de onderstaande 6 fases. Deze fases worden gebruikt om structuur binnen het project aan te brengen.

Fase 1 bestaat uit het definiëren van het project. Hierbij is het de bedoeling om de eisen en wensen van de opdrachtgever te definiëren en aan de hand hiervan requirements voor het project op te stellen.

Fase 2 staat in het teken van onderzoek. Tijdens deze fase wordt de State of the Art bepaald. Verder wordt er gekeken naar mogelijke technieken/methodes die een mogelijke oplossing voor het probleem bieden.

Fases 3 tot en met 5 worden gebruikt voor het iteratief ontwikkelen van een of meerdere prototypes. Wat er tijdens deze fases ontwikkeld wordt hangt af van de eisen/wensen van de opdrachtgever en de resultaten van het onderzoek.

De laatste fase van het project staat in het teken van de afronding van het project. Het gaat hierom afronding van het prototypen en de bijbehorende documenten. Hierbij is het van belang dat alle onderdelen van het project aan de opdrachtgever opgeleverd worden. Ook vindt tijdens deze fase de afstudeer zitting te Fontys plaats.

Overall tijdplan

Fasering	Aantal dagen
1 Project opzet	25
2 Onderzoek	20
3 Iteratie 1	10
4 Iteratie 2	10
5 Iteratie 3	10
6 Iteratie 4	10
7 Afronding	15

Bovenstaande planning kan waar nodig aangepast worden, dit kan gefaciliteerd worden door minder of kortere iteraties te houden.

Fase Project opzet

Omschrijving en aanpak

In deze fase worden de project doelstellingen en planning duidelijk gedefinieerd doormiddel van een PID. Verder worden de eisen aan het project gedefinieerd doormiddel van een MoSCoW document.

Tijdens deze fase zal gewerkt worden aan het definiëren van de onderzoeksopdracht alsmede de eisen aan het project. De PID zal in deze fase aan de hand van feedback van de opdrachtgever en docentbegeleider waar nodig aangepast worden.

Ook wordt deze fase benut om het bestaande project en code base duidelijk te krijgen. Dit wordt gedaan doormiddel van het maken van een UML diagram.

Eindproducten

Aan het eind van deze fase moet de PID goedgekeurd zijn door de opdrachtgever en docentbegeleider.

Startvoorwaarden

N.v.t.

Activiteitenlijst

Activiteit	Aantal dagen
Gespreksformulier	10
PID opstellen	10
Opstellen MoSCoW	5
Eerste bedrijfsbezoek	Eenmalig in week 5

Fase onderzoek

Omschrijving en aanpak

In deze fase wordt onderzoek gedaan naar mogelijke technieken en methodes om de verschillende deelvragen te kunnen beantwoorden. Het doel van deze fase is technieken en methodes identificeren die in de latere iteraties gebruikt kunnen worden.

Tijdens deze fase wordt gebruik gemaakt van bieb- en veldonderzoek zoals beschreven staat in hoofdstuk 1.4.

Deze fase dient als (voor)onderzoek voordat aan de iteraties van prototype(s) begonnen wordt. Dit betekent echter niet dat het onderzoek alleen in deze fase gedaan wordt. Tijdens de hierna komende fases wordt waar nodig ook aan onderzoek gedaan.

Ook kan deze fase benut worden om van gevonden technieken/methodes kleine prototypes te maken. Deze prototypes kunnen gebruikt worden om te bepalen of de gevonden oplossing verder uitgewerkt kan/moet worden tijdens een van de volgende iteraties.

Eindproducten

Deze fase moet aanknopingspunten leveren voor verder onderzoek en ontwikkeling tijdens de stage periode. In overleg met de opdrachtgever wordt bepaald of de gevonden technieken voldoende zijn om een prototypen te ontwikkelen.

Startvoorwaarden

Om deze fase te kunnen beginnen moet de PID met een GO beoordeeld zijn.

Activiteitenlijst

Activiteit	Aantal dagen
Bepalen State of the Art	

Fase Iteratie(s)

Omschrijving en aanpak

De hoeveelheid iteraties en de duur hiervan kan tijdens het project veranderen. Dit wordt bepaald aan de hand van het (voor)onderzoek hierbij is leidend hoeveel mogelijke oplossingen er gevonden zijn. Hierdoor kan het noodzakelijke zijn meer of minder prototypes te realiseren. Ook is het mogelijk dat meerdere iteraties gebruikt worden om het zelfde prototype verder uit te werken. Iedere iteratie heeft een eigen ontwikkelproces bestaand uit 5 fase zoals gedefinieerd in hoofdstuk 1.4.

Voor iedere iteratie wordt aan de hand van de eisen aan het project een doelstelling vastgesteld. Er wordt in overleg met de opdrachtgever bepaald welke gevonden technieken en methodes geselecteerd worden voor de iteratie.

Tijdens de ontwikkeling worden veranderingen aan de bestaande code base gedocumenteerd met behulp van een UML document. Verder worden keuzes die genomen zijn tijdens de ontwikkeling gedocumenteerd.

De prototypes worden ontwikkeld om gevonden technieken en methodes te kunnen valideren voor het onderzoek en zo de beste mogelijke oplossing te vinden.

Eindproducten

De eindproducten binnen de specifieke iteraties worden aan de hand van het MoSCow document en de specifieke eisen/wensen van de opdrachtgever bepaald.

Startvoorwaarden

Het onderzoek moet ver genoeg gevorderd zijn om een prototype te kunnen realiseren van 1 of meer gevonden technieken/methodes. Ook moet het gemaakte MoSCoW document door de opdrachtgever goedgekeurd zijn.

Activiteitenlijst

Activiteit	Aantal dagen
Iteratie doelstelling formuleren	1
Prototypen ontwerpen	1
Prototypen maken	8
Aanpassingen documenteren	Waar nodig
Strandup	Dagelijks
Voortgang bespreken	Wekelijks

Fase Afronding

Omschrijving en aanpak

Deze fase wordt gebruikt om de stage periode af te ronden bij zowel het stage bedrijf als de opleiding.

Hierbij gaat het om de afronding en oplevering van documenten en prototypes. Ook vindt in deze fase de afstudeerzitting te Fontys plaats.

Eindproducten

Het eind product van deze fase is een scriptie die gebruikt wordt om de afstudeer stage te beoordelen.

Ook worden de eventuele ontwikkelde prototypes en andere bevindingen opgeleverd aan het bedrijf.

Verder wordt een architectuur document opgeleverd aan het stage bedrijf als ondersteuning van een eventueel prototype wat ontwikkeld is.

Startvoorwaarden

N.v.t.

Activiteitenlijst

Activiteit	Aantal dagen
Opleveren/afronden afstudeerscriptie	5
Opleveren/afronden prototype(s)	5
Opleveren/afronden documentatie	5
Tweede bedrijfsbezoek	Eenmalig in week 16/17

Kwaliteitsbewaking, Testen, Configuratiemanagement

Goedkeuringen

Wat	Wie	Wanneer
1 PID	Paul Brinkkemper Mark van Kuijk	Week 5
2 MoSCoW	Paul Brinkkemper	Week 5
3 Proof of concept(s)	Paul Brinkkemper	Tussentijds Week 5 - 19
4 Afstudeer Scriptie	Mark van Kuijk Luuk Waarbroek Paul Brinkkemper	Week 19/20
5 Architectuur document	Paul Brinkkemper	Week 19/20
6 Prototype	Paul Brinkkemper Mark van Kuijk	Week 19/20

Testaanpak/strategie

Ontwikkelde prototypes worden binnen het bedrijf getest en beoordeeld op de bruikbaarheid voor het eindproduct. Het gaat hier om functionele testen die moeten bepalen of er voldaan wordt aan de te realiseren functionaliteit.

Reviews

Iedere iteratie wordt afgesloten met een sprint review. Deze review wordt gebruikt om het gebruikte proces te evalueren en eventuele verbeterpunten te formuleren.

Ook wordt deze review gebruikt om de doelstelling van de komende iteratie te bepalen.

Risico's en afhankelijkheden

Afhankelijkheden

Het afstudeerproject is niet afhankelijk van andere projecten/resources binnen het bedrijf.

Projecten die van dit project afhankelijk zijn

MoneyMaker Deluxe kan zonder het resultaat van het afstudeerproject gewoon verder ontwikkeld worden. Hierdoor is het wel mogelijk dat de ontwikkeltijd van het project omhoog gaat.

Risico's en uitwijkactiviteiten

Risico	Activiteiten ter voorkoming opgenomen in plan	Uitwijkactiviteiten
1 No-Go advies in week 5.	Tijdig de PID en opdracht laten valideren door de docentbegeleider	In overleg met docentbegeleider zoeken naar een mogelijke oplossing om alsnog een GO advies te krijgen.
2 Raak project kan/heeft verwachtingen/eisen aan het project.	Project doelen afstemmen met opdrachtgever en waar mogelijk met het raak project. Waarbij eisen en wensen van de opdrachtgever leidend zijn.	De voltooiing van het project wordt beoordeeld door het opleidingsinstituten en de opdrachtgever. Het Raak project neemt hier geen deel in.
3 De te ontwikkelen features zijn niet complex genoeg als afstudeer opdracht.	Doormiddel van een MoSCoW document vroeg in het proces de afstudeer opdracht laten valideren door de docentbegeleider.	In overleg met de docentbegeleider en bedrijfsbegeleider bepalen hoe de opdracht alsnog complex genoeg gemaakt kan worden.

Bijlage II: Tips & Tops

Tops

Methodisch en gestructureerd

Ik probeer waar het ondersteund wordt door het bedrijf/team te werken doormiddel van SCRUM/Agile. Ikzelf vindt het gebruik van SCRUM/Agile erg handig omdat het mij een goed beeld geeft van wat ik nog allemaal moet realiseren binnen een spel. Dit zorgt er voor dat ik tijdens het ontwikkelen zo veel mogelijk rekening kan houden met eventuele andere functionaliteit die gebruik moet maken van classes/methodes die ik maak.

Het niet gebruik maken van deze technieken zorgt er bij mij persoonlijk voor dat ik het overzicht verlies binnen het project waardoor ik soms dingen overnieuw of anders moet doen.

Tijdens mijn stage periode ontbrak het aan ontwikkelmethodiek en backlog, hierdoor was het vaak onduidelijk wat er gemaakt moest worden en wat al gedaan was. Om deze problemen op te lossen heb ik zelfstandig een backlog gemaakt en ben ik deze gaan gebruiken als ondersteuning voor Agile werkomgeving met sprints van 2 weken.

Samenwerken

Tijdens verschillende projecten, vakken en stages heb ik uitgebreid moeten samenwerken met verschillende mensen en disciplines.

Samenwerken met verschillende disciplines vindt ik heel erg leuk om te doen, dit omdat iedereen iets unieks toevoegt aan het project. Door de verschillende invloeden van de verschillende disciplines wordt het project

Tijdens verschillende stage en projecten heb ik samengewerkt in teams met verschillende disciplines waaronder Software Engineering, Media, Marketing, Creative/Artist en ACI.

Generalist

Tijdens de Minor heb ik in samenwerking met een aantal studenten van verschillende opleidingen een Escape Room ontwikkeld. Tijdens dit project hebben we samengewerkt met Game Design & Technology, Cyber Security, Marketing, Communicatie en Technology. Hierbij hebben we ons best gedaan de taken zo goed mogelijk te verdelen zodat deze goed aansloten op de betreffende specialisaties.

Tips

Communicatievaardigheden

Tijdens mijn laatste stageperiode ben ik zelfstandig verantwoordelijk geweest onder andere de planning en de algehele ontwikkeling (ik was de enigste programmeur). Hierdoor moest ik veel vergaderen met het management over voortgang en nieuw te ontwikkelen functionaliteit.

In het verleden heb ik gemerkt dat ik duidelijker moet overbrengen waarom bepaalde dingen naar mijn mening niet kunnen, hier betreft het vaak dingen die wel mogelijk zijn maar te lang duren om binnen de sprint/andere deadlines te passen.

Kritische Houding

In de voorgaande stage periode heb ik een aantal keer het probleem gehad dat functionaliteit niet volledig voldeed aan de eisen van de opdrachtgever. Hierdoor heb ik een aantal keer extra tijd moeten besteden om dit te repareren.

Dit kan ik in de toekomst voorkomen door beter door te vragen naar wat de functionaliteit moet doen. Wat tijdens de stage ook veel hielp was een korte omschrijving maken zoals ik dacht dat het gemaakt moest worden en dit laten goedkeuren en eventueel aanpassen door de opdrachtgever.

Onderbouwing en verantwoording

Tijdens mijn voorgaande stage en onderzoeks vakken heb ik zelfstandig een onderzoek met artikel moeten realiseren. Hierbij heb ik in het verleden problemen gehad met het duidelijk overbrengen waarom er voor een bepaalde oplossing gekozen was.

Dit gaat voor mijn een belangrijk focus punt worden tijdens mijn afstudeer stage, juist om ervoor te zorgen dat ik een duidelijk en goed verslag maak. Om dit te ondersteunen wil ik in overleg met mijn stage begeleider een aantal keer mijn verslag nakijken en mogelijke verbeterpunten formuleren.

Verantwoording

Originaliteit en creativiteit

Tijdens mijn voorgaande stage was ik de enigste programmeur aan een project waarbij ik samenwerkte met een aantal andere stagiaires(2D art, 3D art, marketing). Tijdens het project was ik verantwoordelijk voor alle nieuwe ontwikkeling. Tijdens de ontwikkeling was ik verantwoordelijk voor het ontwerpen en realiseren van nieuwe functionaliteit.

Informatievaardig

Tijdens mijn stage heb ik in overleg met de opdrachtgever en andere partijen een complete lijst met gewenste functionaliteit opgesteld. Dit doormiddel van gesprekken/vergaderingen en het afstemmen van de verschillende aspecten.

Reflectievermogen

Tijdens de opleiding is een van mijn focus punten het verbeteren van communicatie en presentatie geweest. Tijdens zowel de stage presentatie als de stage zelf heb ik positieve feedback gehad over de manier waarop ik communiceerde en presenteerde.

Ontwikkelingsgericht

Ik probeer tijdens projecten zoveel mogelijk dingen te doen waar ik nog geen eerdere ervaring mee heb. Dit om kennis te maken van meer verschillende technieken en methodes die binnen mijn vakgebied vallen.

Bijlage III: Klasse diagram systeem

Klasse diagram Money Maker Deluxe

Bestaande project

Hier worden de klassen weergegeven en omschreven die direct te maken hebben met het economische systeem van Money Maker Deluxe. In het hoofdstuk Relaties is een overzicht te vinden van de onderlinge relaties tussen de klassen.

Trustee - Trustee is de basis klasse voor alle in game entiteiten hiervoor wordt gebruik gemaakt van overerving. Trustee bied de basis functionaliteit voor Handel, Leningen en Visualisatie aan.

Trustee	<div>+ resourcesFlow : Trustee</div> <div>+ matClient : Material</div> <div>+ matSupplier : Material</div> <div>+ priceCalculationDay : int</div> <div>+ purchaseCalculationDay : int</div> <div>+ ExecuteTrusteeEvent(te : TrusteeEvent, trustee : Trustee) : void</div> <div>+ Awake() : void</div> <div>+ Init() : virtual void</div> <div>+ myAwake() : virtual void</div> <div>+ myStart() : void</div> <div>+ Update() : void</div> <div>+ OnDestroy() : void</div> <div>+ CheckLeave() : void</div> <div>+ CollectCoin() : void</div> <div>+ ChangePriceOfOwnedProperty(price : float) : void</div> <div>+ LetManagerDecide() : void</div> <div>+ GetCurrentPrice() : void</div> <div>+ PriceChange(_change : bool) : void</div> <div>+ DayStartEvent(dayCount : int) : virtual void</div> <div>+ DayEvent(daysCount : int) : virtual void</div> <div>+ MonthStartEvent(monthCount : int) : virtual void</div> <div>+ UpdateMonthlyTrades() : void</div> <div>+ compareLists(list1 : List<TradeContract>, list2 : List<TradeContract>) : bool</div> <div>+ MonthEvent(count : int) : virtual void</div> <div>+ YearEvent(count : int) : virtual void</div> <div>+ SellsProduct(_productType : int) : bool</div> <div>+ BuysProduct(_productType : int) : bool</div> <div>+ GetDemandForProduction(_productType : int) : int</div> <div>+ GetSupply(_product : int) : int</div> <div>+ GetPrice(_product : int) : float</div> <div>+ GetCashAmtInTrustee() : virtual float</div> <div>+ AddCashAmtInTrustee(amount : float) : virtual void</div> <div>+ GetUnsoldProductsAmount() : int</div> <div>+ GetActualUnsoldProductsAmount() : int</div> <div>+ GetSoldProductAmount() : int</div> <div>+ GetSoldProductAmount(ofProduct : int) : int</div> <div>+ GetRevenue() : float</div> <div>+ GetRevenue(ofProduct : int) : float</div> <div>+ GetPurchasedProductsAmount() : int</div> <div>+ GetPurchasedProductsAmount(ofProduct : int) : float</div> <div>+ GetCost() : float</div> <div>+ GetCost(ofProduct : int) : float</div> <div>+ GetAverageCost(ofProduct : int) : float</div> <div>+ GetDayProfit() : float</div> <div>+ EstimateDayProfit() : float</div>	<div>+ GetActualDayProfit() float</div> <div>+ GetAmountOfProduction() : float</div> <div>+ GetPrevMonthProfit() : float</div> <div>+ CanLinkSale(contract : TradeContract) : bool</div> <div>+ LinkSale(contract : TradeContract) : void</div> <div>+ CanLinkPurchase(contract TradeContract) : bool</div> <div>+ LinkPurchase(contract : TradeContract) : void</div> <div>+ UnlinkContract() : void</div> <div>+ AdjustDemand(_tradeContract) : int</div> <div>+ AdjustSupply(_product : int, withAmount : int) : int</div> <div>+ AdjustPrice(ofProduct : int, withFactor : float, withAmount : float) : float</div> <div>+ CanBuyProducts(contract : TradeContract) : bool</div> <div>+ CanSellProducts(contract : TradeContract) : bool</div> <div>+ AdjustCash(amount : float) : void</div> <div>+ GoOutOfBusiness() : void</div> <div>+ DemandLoan() : void</div> <div>+ GiveLoan(loan : Loan) : void</div> <div>+ IncreaseLoan(amount : float, loan : Loan) : void</div> <div>+ EndDemandLoan() : void</div> <div>+ HasLoans() : bool</div> <div>+ HasDeposits() : bool</div> <div>+ NeedsLoan() : bool</div> <div>+ NeedsDeposits() : bool</div> <div>+ GetLoanTo(borrower : Trustee) : Loan</div> <div>+ GetLoanFrom(lender : Trustee) : Loan</div> <div>+ IssueCreditLoan(loan : Loan) : virtual void</div> <div>+ Pay(amount : float) : virtual void</div> <div>+ AcceptSavings(loan Loan, wallet : Wallet) : virtual bool</div> <div>+ ReceivePayment(wallet : Wallet) : virtual bool</div> <div>+ CanDoPayLoanPayment(loan : Loan) : virtual bool</div> <div>+ PayLoanPayment(loan : Loan) : virtual void</div> <div>+ RepayLoanTerm() : virtual void</div> <div>+ DefaultLoanPayment(loan : Loan) : virtual void</div> <div>+ RemoveLoanFromBooks(loan Loan) : virtual void</div> <div>+ GetProducingProduct() : int</div> <div>+ ShowDependencies() : void</div> <div>+ DisplayDependencies() : virtual void</div> <div>+ HideDependencies() : virtual void</div> <div>+ Highlight(isClient : bool) : void</div> <div>+ HideHighlight(isClient : bool) : void</div> <div>+ UpdateFlows() : void</div> <div>+ SpawnFlow(source : Trustee, target : Trustee) : List<GameObject></div> <div>+ SpawnFlowProducts(flowCubes : List<GameObject>, product : int) : void</div>
---------	--	--

Company : Trustee
+ Product : int + img : Sprite + tasks : CompanyAltasks + behaviourTree : BehaviourTree + UpgradeNeeds : List<ProductSet> + currentUpgradeTime : int + UpgradeTime : int + UpgradeLevel : int + UpgradeMultiplier : float + upgrader : CompanyUpgrader + pastPrice : float + maximumPriceBuying : float + minimumPriceSelling : float + busted : bool + bustTime : float + upgrading : bool + averageProfit : float + totalProfit : float + monthsPast : float
+ Init() : override void + OnEnable() : void + Reset() : void + IsMakingProfit() : bool + setBust(value : bool) : void - Produce(dayCount : int) : void + getProductionState() : bool + StartProduction() : void + HoldProduction() : void + UpgradeNoLoan() : void + UpgradeCompanyWithLoan(loan : Loan) : void + GetBuildingPrice() : float + CalculateUpgradePrice(shouldBeAvailable : bool) : float + CanDoPayLoanPayment(loan : Loan) : override bool + DayStartEvent(dayCount : int) : override void + DayEvent(daysCount : int) : override void + MonthEvent(count : int) : override void

Company – Company overerft van Trustee en is verantwoordelijk voor het gedrag van een Bedrijf binnen het spel. De functionaliteit van deze klasse richt zich voornamelijk op het beslissingen proces van een bedrijf en de acties die het bedrijf kan ondernemen.

Wallet – Wordt gebruik door Entiteiten om verschillende soorten Geld (Goud, Krediet, Valuta) op te slaan. Deze klasse bevat methodes om het geld te tellen, valideren dat een betalingsmiddel geaccepteerd wordt en het verwijderen en ontvangen van betalingen.

Wallet
+ MinimumTrustLevel : int + Moneys : List<Money>
+ Wallet() : void + Count() : void + Count(gold : float, credits : float) : void + CountCredit() : float + CountGold : float + CanPay(amount : int) : bool + CanPay(amount : float, withMinimalTrustLevel : int) : bool + Pay(amount : float, minimalTrustLevel : int) : Wallet + RemoveFromWallet(amount : float, minimalTrustLevel : int) : Money + AcceptsCredit(fromTrustees : List<Trustee>) : bool + AcceptsCredit(fromTrustee : Trustee) : bool + AcceptsCredits(fromWallet : Wallet) : bool + ReceivePayment(wallet : Wallet) : void + ReceivePayment(money : Money) : void

Money – Deze klasse slaat gegevens over een geld bedrag op.

Money
+ type : MoneyType + Amount : float + trustee : Trustee
+ Money(amount : float) : void + Money(amount : float, trust : Trustee) : void

Market – Word gebruikt door de bedrijven om onderling te kunnen handelen. Houd alle verkochte en geproduceerde goederen bij.

Market
<ul style="list-style-type: none"> + OutbiddingFactor : float + PriceAdjustmentAmount : float + ProductGroups : List<ProductGroup> + AllInhabitants : List<Trustee> + TrusteeDemandingLoans : List<Trustee> + LoansInDefault : List<Loan> + dataStorages : List<PriceDataStorage> + GDP : float + CPI : float + CPIIndexation : List<float> + GDPIndexation : List<float> + isIndexing : bool + globalMarket : WorldMarket + marketName : string + firstBuilding : Transform + Wallet : Wallet + productInfoToDisplay : List<ProductInfoToDisplay> + mouseHover : bool + GUIManagerScript : GUIManager
<ul style="list-style-type: none"> + Start() : void + Initialize(buttons : List<Button>, worldMarket : WorldMarket) : void + Update() : void + OnEnable() : void + OnDisable() : void + RemoveBrokeTrustees() : void + HoldProduction(_me : Trustee) : void + StartProduction(_me : Trustee) : void + AddNewCompanyEntry() : void + RemoveTrustee(toRemove : Trustee) : void + AddInhabitantToMarket(trust : Trustee) : void + AddDemand(demander : Trustee, forProduct : int) : void + AddSupply(supplier : Trustee, forProduct : int) : void + AddSale(sale : TradeContract) : void + RemoveDemand(consumer : Trustee, forProduct : int) : void + RemoveSupply(supplier : Trustee, forProduct : int) : void + RemoveSale(sale : TradeContract) : void

<ul style="list-style-type: none"> + FillDemand(consumer : Trustee, forProduct : int, outbid : bool) : void + SellSupply(seller : Trustee, forProduct : int, firesale : bool) : void + GetTrusteeOfType(companyType : CompanyType) : List<Trustee> + GetAverageMarketProfit(set : ProductSet) : float + GetAverageMarketPrice(need : ProductSet) : float + GetAverageMarketNegotiatedPrice(buyer : Company, need : ProductSet) : float + GetCheapestCompetitor(theTrusteeAsking : Trustee) : Trustee + GetAvailableMarketPrice(need : ProductSet, forBuyer : Trustee) : float + GetUnfilledDemand(forProduct : int) : int + GetSellSmallerThenDemand(_product : int) : bool + GetUnsoldSupply(forProduct : int) : int + DayStartEvent(dayCount : int) : void + DatEvent(dayCount : int) : void + QuarterMonthEvent(count : int) : void + MonthEvent(count : int) : void + YearEvent(count : int) : void + SetMarketDta(_productGroup : ProductGroup, totalAmountSold : int, avgPriceOfProduct : float) : void + AddToDataStorage(index : int, totalAmountSold : float, storedSupply : int, totalDemand : int) : void + GetIndexation(valueList : List<float>) : float + MakeIndexation(indexList : PriceDataStorage, value : float, index : float) : float + RemoveSales(productGroup : ProductGroup) : void + GetSalesAmount(sales : List<TradeContract>) : int + GetTotalDemand(demanders : List<Trustee>) : List<int> + GetSupplyStored(suppliers : List<Trustee>) : List<int> + GetTotalProductionForProduct(prod : int) : int + GetTotalDemandForProduct(prod : int) : int + GetTotalAmount(set : List<ProductSet>, totalAmount : List<int>) : List<int> + GetAmountOfCompanies(product : int) : int + GetProperties(player : Player) : List<Company> + CheckCompanyLeaving() : void + SpawnNewCompany() : void + DemandLoan() : void + CancelDemandLoan(forTrust : Trustee) : void + LoanTermEvent(loan : Loan) : void + LoanDefaultingEvent(loan : Loan) : void + LoanDelinquencyHandledEvent(loan : Loan) : void + LoanRepayEvent(loan : Loan) : void + UpdateBankConnections(loan : Loan) : void + LoanForgivenEvent(loan : Loan) : void + RecalculateGUIMarketPrice() : void
--

TradeContract – Een overeenkomst tussen bedrijven om bepaalde goederen te verkopen aan elkaar.

ProductGroup – Een overzicht van alle bedrijven die bij een Market horen die een bepaald product nodig hebben.

ProductGroup
+ Product : Product + Demand : List<Trustee> + OldDemand : List<Trustee> + UnfulfieldDemand : List<Trustee> + Supply : List<Trustee> + OldSupply : List<Trustee> + UnfulfieldSupply : List<Trustee> + RemoveSupplies : List<Trustee> + RemoveDemands : List<Trustee> + Sales : List<TradeContract> + AveragePrice : float + PriceData : DataStorage + compNegotiation : CompanyNegotiator
+ ProductGroup(forProduct : Product) : void + Destroy() : void + DayStartEvent() : void - QuarterMonthEvent(count : int) : void - MonthEvent(count : int) : void - YearEvent(count : int) : void + GetAveragePrice() : float + GetAveragePrice(amount : int) : float + GetAverageNegotiatedPrice(buyer : Company, amount : int) : float + GetAvailableMarketPrice(amount : int, forConsumer : Trustee) : float + GetUnfulfilledDemand() int + GetUnsoldSupply() : int + GetSaleVolume() : int + AddSupply(newSupply : Trustee) : void + AddDemand(newDemand : Trustee) : void + AddSale(newContract) : void + RemoveSupply(oldSupply : Trustee) : void + RemoveDemand(oldDemand : Trustee) : void - ClearOldInfo() : void + RemoveOld() : void + SupplySmallerThenDemand() : bool + SupplyMinusDemand() : int + RemoveSale(oldContract : TradeContract) : void - fillAllDemand() : void + FillDemand(consumer : Trustee, outbid : bool) : void + SellSupply(seller : Trustee, firesale : bool) : void - calculateAveragePrice() : float

TradeContract
+ Supplier : Trustee + Client : Trustee + Product : Product + Amount : int + PricePerUnit : float + ContractPrice : float
+ TradeContract(_client : Trustee, _supplier : Trustee, _product : Product, _amount : int) : void + CheckContract() : void + ExecuteContract() : void + Cost() : float

ProductSet – Een grondstof en hoeveelheid die een bedrijf nodig heeft. Wordt gebruikt voor Upgrades en Productie kosten.

ProductSet
+ Amount : int + MaxAmount : int + UnitPrice : float + Product : Product
+ ProductSet(_product : Product, _amount : int, _unitPrice : float) : void + CompareProduct(other : Product) : bool + CompareAmount(_amount : int) : bool + Remove(amount : int) : bool

Enums – Naast de afgebeelde klassen zijn er de volgende Enums:

- LendingStage
- OwnerType
- TerrainType
- MoneyType
- Product
- CompanyType
- RelationshipStatus
- RawMaterialType
- RiskType
- TilesLayout
- Industry

<<Enum>> LendingStage Gold Credit FReserve	<<Enum>> MoneyType Credit Gold	<<Enum>> CompanyType None House Mill Farm Smith Mine Woodcutter Trader Player Bank	<<Enum>> RelationshipStatus None Debtor DelinquentDebtor Depositor Property Restricted	<<Enum>> RawMaterialType None Forest Grassland Rocks Swamp Water
<<Enum>> OwnerType PriceStable PriceDecreasing PriceIncreasing Undefined	<<Enum>> Product None Labour Bread Wheat Tool Ore Wood Credit		<<Enum>> Industry Bread Tools Wood	<<Enum>> TilesLayout Hex Square_4 Square_8
<<Enum>> TerrainType Normal Forest Water Mountain Impassable				<<Enum>> RiskType Risky Regular Safe

Player : Trustee
+ GUIManagerScript : GUIManager + canUpgrade : bool + UpgradeNeeds : List<ProductSet> + upgrading : bool + hadDepositTutorial : bool + monthlyCreditCalls : float + monthIncome : float + monthOutcome : float + monthProfit : float + playerBanks : List<Bank> + playerName : string
+ Start() : void + GetBankInMarket(market : Market) : Bank + UpgradeBank() : void + GetMaxConnections() : int + GetCurrentConnections() : int + UpgradeBankLevel(bank : Bank) : void + UnlockUpgrade() : void + IssueCreditLoan(loan : Loan) : override void + AcceptSavings(loan : Loan, wallet : Wallet) : override bool + ReceivePayment(wallet : Wallet) : override bool + RemoveLoanFromBooks(loan : Loan) : override void + CanDoLoanPayment(loan : Loan) : override bool + PayLoanPayment(loan : Loan) : override void + MonthEvent(count : int) : override void + UpdateFinanceView() : void + GetBankGUIInformation() : BankGUIInformation + DayEvent(daysCount : int) : override void - CalculateCallbackChance() : float - DoCallback(bank : Bank) + Bankrupt() : void - Explode() : void - SendRioters() : void - CreateRioter(type : CompanyType, bank : Bank) : void + ExpectedIncome() : float + ExpectedExpenses() : float

Storage – Maakt het mogelijk meerdere productsets op te slaan en te bewaren.

Storage
- _Storage : List<ProductSet>
+ AddProductToStorage(_product : Product, _amount : int, _max : int) : void + AddAmountToStorage(_product : Product, _amount : int) : int + SetProductLimit(_product : Product, _max : int) : void + SetProductLimit(_product : Product, _max : int) : void + CheckProductSpaceMac(_product : Product) : bool + CheckIfStorageFull() : bool + CheckProductSpaceMax(_productSet : ProductSet) : bool + CheckProductAvailableMin(_product : Product) : bool + RemoveSpecificAmountFromStorage(_product : Product, _amount : int) : int + RemoveAllAmountFromStorage(_product : Product) : ProductSet + RemoveProductFromStorage(_product : Product) : void + SetProductPrice(_product : Product, _price : float) + GetProductPrice(_product : Product) : float + GetAmount(_product : Product) : int + GetAmountSpaceLeft(_product : Product) : int + FindProductInStorage(_product : Product) : ProductSet + FindProductIndexInStorage(_product : Product) : int + GetTotalAmountOfProductsStorage() : int

Player – Deze klasse overerft van Trustee en houdt de gegevens en acties van de speler bij.

Loan – Slaat de gegevens van een specifieke lening op.

CompanyUpgrader – Verantwoordelijk voor het uitvoeren van een upgrade van een bedrijf.

CompanyUpgrader
- company : Company + isUpgrading : bool
+ UpgradeProductionCompnay() : void + AddUpgradeNeeds() : void + UpgradeCompanyWithLoan(loan : Loan) : void - CanPayForUpgrade() : bool + CheckUpgrade() : void

CompanyNegotiator – Verantwoordelijk voor het aankopen van producten voor een bedrijf. Dit kan voor zowel productie als een upgrade zijn.

CompanyNegotiator
+ prodGroup : ProductGroup
+ FillAllDemand() : void + SortTrusteesOnPrice(trustees : List<Trustee>, isMaximumPrice : bool) : List<Trustee> + ConnectTrustee(consumer : Trustee) : List<TradeContract> + NegotiateContractPrice(contract TradeContract) : void + CalculateContractPrice(seller : Company, buyer : Company, productAmount : int, product : Product) : float + CalculateSupply(prod : Product) : int + CalculateDemand(prod : Product) : int

Bank – Begin klasse om AI banken mee te ontwikkelen. Functionaliteit is nog niet werkend.

Bank : Trustee
+ owner : Player + upgradeLevel : int + maxConnections : int + currentConnections : int + upgradeConnections : int[] + explosionAnim : Animator + fireAnim : Animator
+ Init() : override void + GetCashAmtInTrustee() : override float + AddCashAmtInTrustee(amount : float) : override void

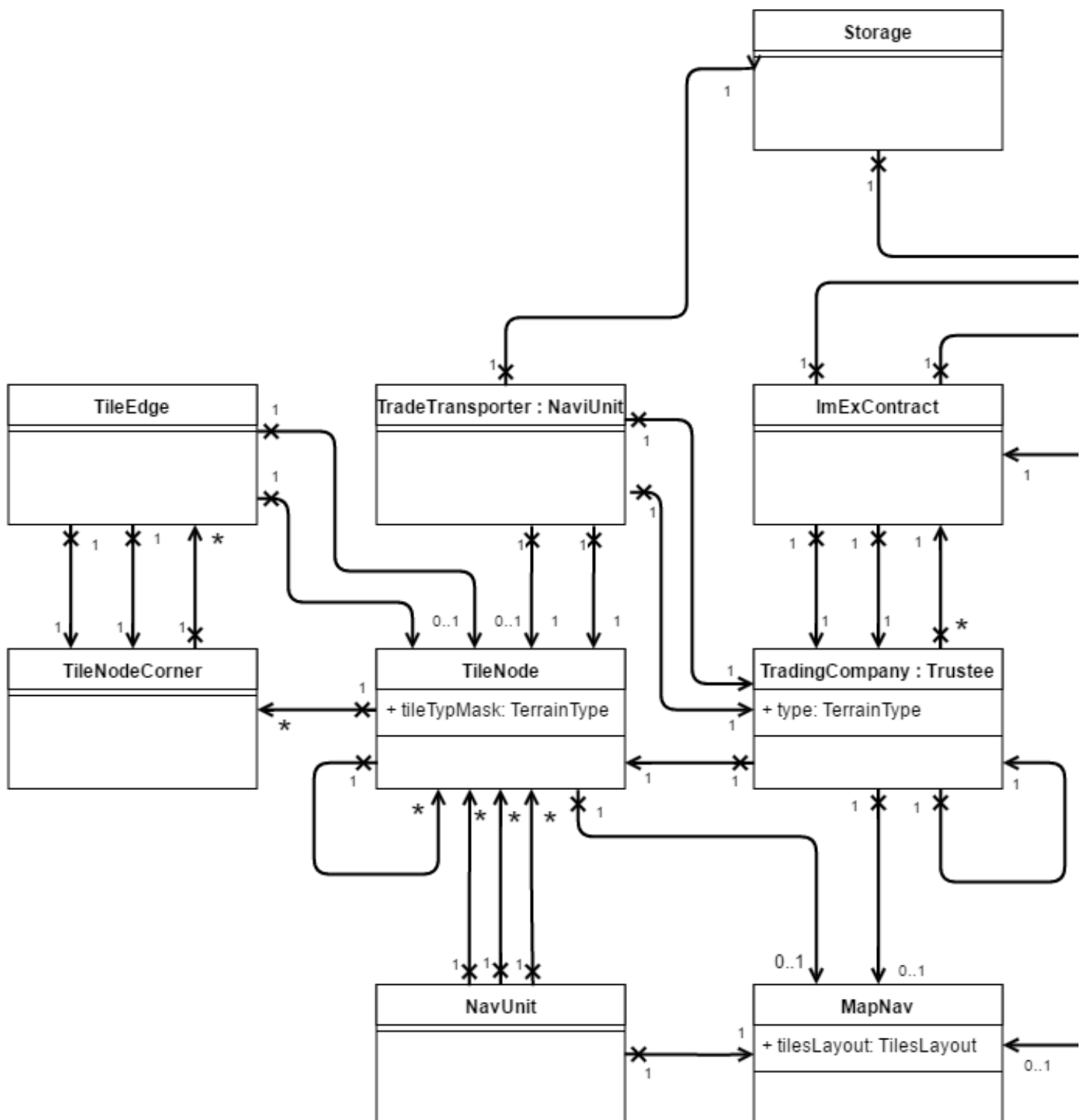
Loan
+ Principal : int + Interest : int + Lender : Trustee + Borrower : Trustee + PaymentTerms : int + CurrentTerm : int + DaysPerTerm : int + PaymentDate : int + PaymentPerTerm : float + RepayedAmount : float + Defaulting : bool + contract : ImExContract + LoanStarted : LoanAction + LoanTermPayed : LoanAction + LoanTermDefaulting : LoanAction + LoanDefaultingHandled : LoanAction + LoanRepayedFull : LoanAction + LoanForgivenFull : LoanAction + laonAmount : int
+ ExecuteLoanAction(la : LoanAction, loan : Loan) : void + Loan(principal : int, interest : int, lender : Trustee, borrower : Trustee, paymentterms : int, daysperterm : int, contract : ImExContract) : void + CreateSavingsLoanToBank(loan : Loan) : bool + setDaysPerTerm(_DaysPerTerm : int) : void + getDaysPerTerm() : int + CreateCreditLoanToCompany(loan : Loan) : bool + PayTerm() : void + CannotPayTerm() : void + LoanFullyRepaid() : void + LoanForgiven() : void + LoanNegotiated() : void + ToString() : override void + GetTotalReturn() : int + GetTotalReturnLeft() : float

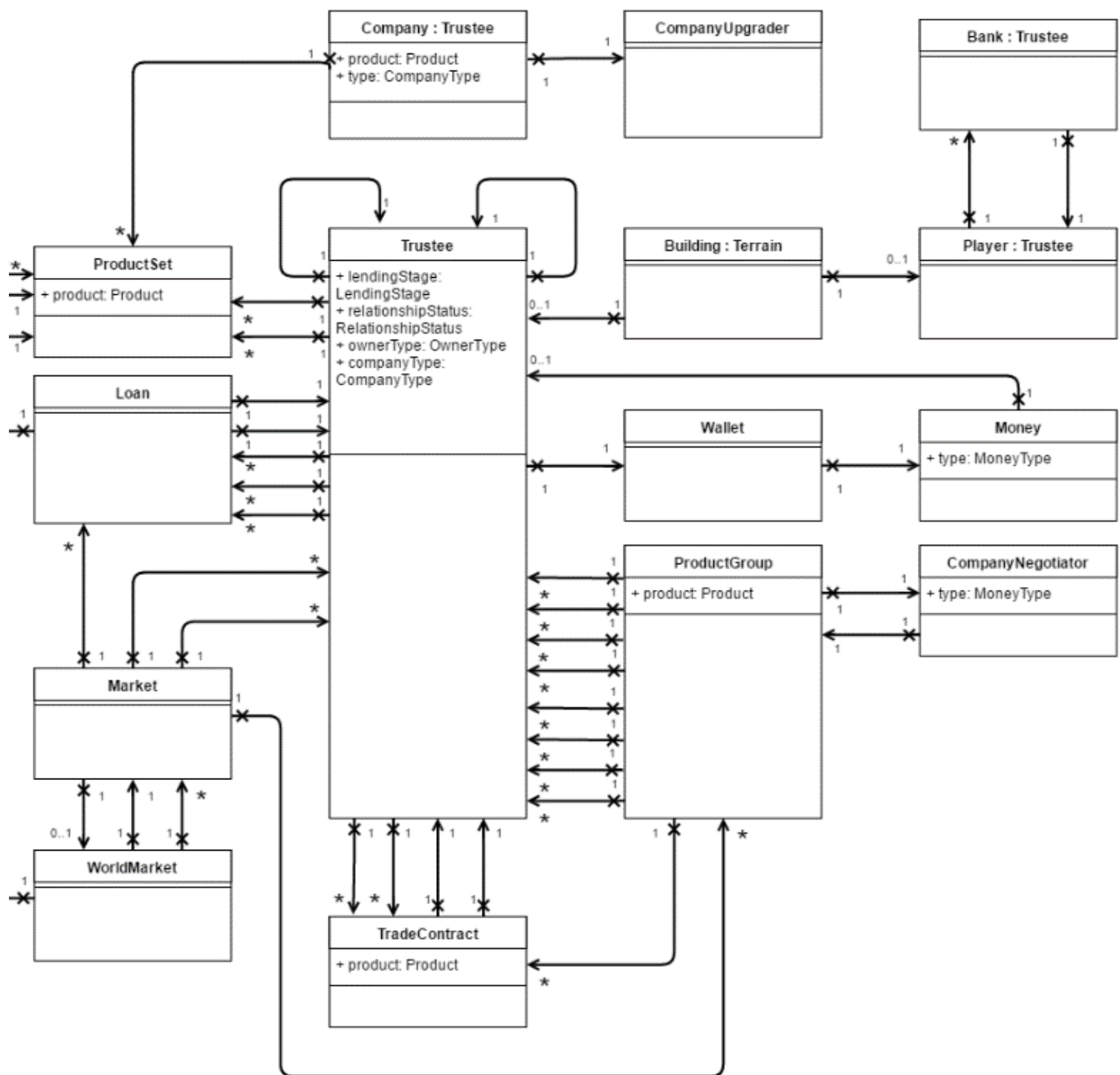
De volgende klassen worden hier niet afgebeeld omdat deze niet langer gebruikt worden in het project:

- TradingCompany
- StorageItem
- ImExContract
- ArtificialBanker

Verder is er in het project een basis opzet voor een Behaviour tree gemaakt. Deze wordt gelinkt aan de huidige Company klasse. Helaas is deze implementatie nooit afgerond en is deze helaas momenteel niet bruikbaar.

Relaties





Klassen die in het hoofdstuk hierboven niet uitgebreider beschreven worden hebben geen of weinig invloed op het economische systeem van Money Maker Deluxe. Dit zijn klassen die voornamelijk de wereld en het uiterlijk/interactie hiervan regelen. Deze hebben geen tot weinig impact op het nieuw te ontwikkelen systeem en zijn daarom niet verder uitgewerkt.

Overige scripts

Hier volgt een overzicht van alle klassen die niet opgenomen zijn in het bovenstaande klasse diagram. Deze zijn niet opgenomen omdat deze geen directe relatie hebben met het economische systeem zoals dat in het klasse diagram afgebeeld wordt.

De meeste klassen die in dit overzicht weergegeven worden zijn “single responsibility” klassen. Veel van deze klassen worden gebruikt om 1 specifiek in game UI onderdeel weer te geven.

Omdat deze klassen geen directe relatie hebben met het economische systeem is ervoor gekozen hiervan geen klasse diagram te maken omdat er geen of weinig aanpassingen gedaan hoeven te worden aan deze code.

Dikgedrukte text staat voor Directories, tabs worden gebruikt om sub files/directories weer te geven.

Editor

AlphaNumericSort
FindMissingScriptsRecursively
GUIHierarchy
MMTileNodeEditor
SwitchMenuGameEditorScript
TutorialGUIEditor

GUI

ClickOutsideGui

GUIPanels

ActivityGUI
ActivityLog
BankScrollViewItem
BankUpgradeScreen
BaseTileGUI
ExtraDetailsOverview
FinanceOverview
FutureTransactionOverviewGUI
FutureTransactionView
GraphView
GUIPanelHandler
InGameMenuController
InspectLoanGUI
LineGraphView
MessageBoxGUI
MessageHandler
NewBuildingLoanGUI
NewLoanGUI
ProfitPredictorGUI
RecoupLoanGUI
SavingsOfferdGUI
TileInvestment
TutorialGUI
VaultScreen

HoverPanel

CompanyDisplayer
HoverPanelResultRow

InGameIcons

CoinCollectFeedback
ForwardMonth
IconLoader
PopuIcon

Others

AnimationsLoader
ChangeShader
CloudParticles
GraphLabel
GUIManager
LoanGraph
MarketNameDisplay
MarketNameGenerator
ObjectHighlighter
OptionView
ProductListView
ProductView
SelectedTileView
SpriteLoader
TextLoader
TileGUI
TileIndicators
Tooltip
TutorialLoader

LevelEditor

CameraController
DataManager
EditorMenuScript
GoalScript
GoalTracker
MapBuilder
MMEditorController
ToolScript

UIElements

UIFileMenu
UINewMap
UISelection
UITerrainBrush
UITileProperties
UITrusteeProperties

Terrain

TerrainLoader

Units

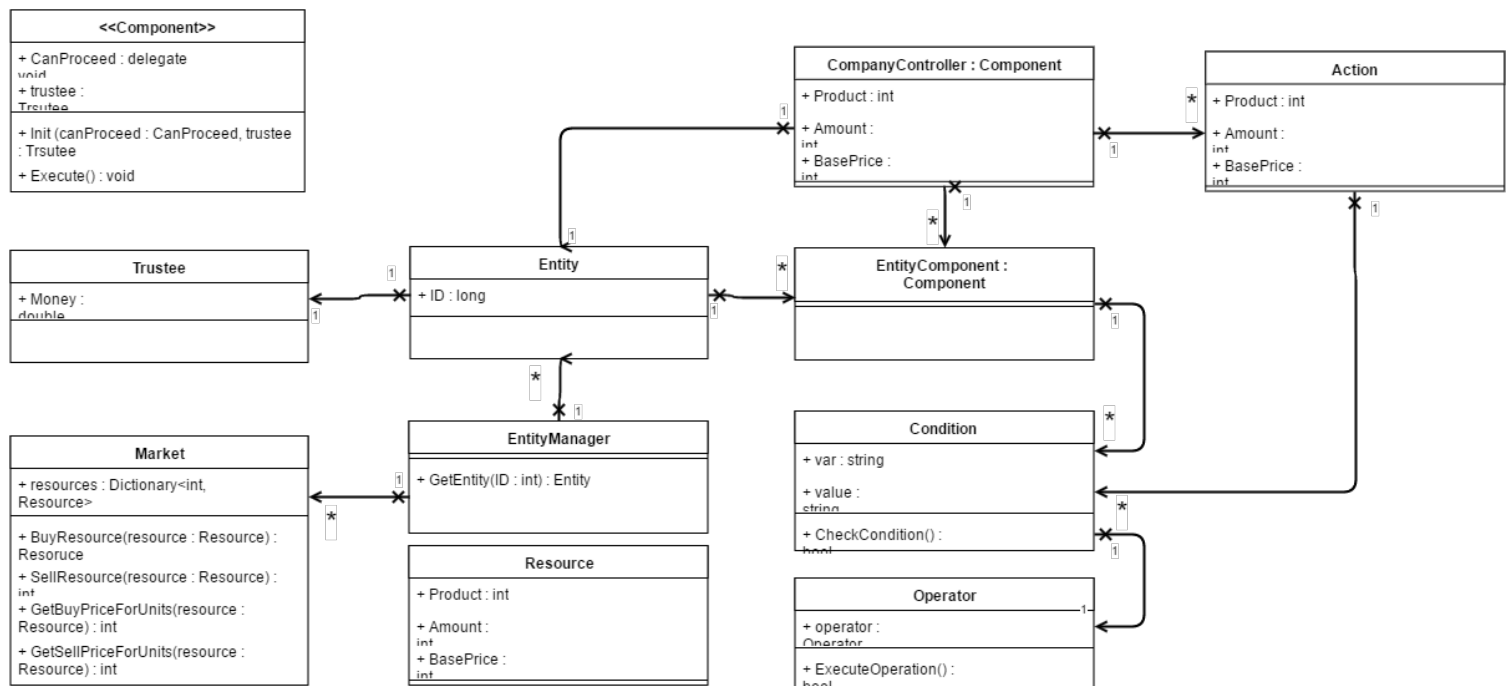
VisualHandler

Utils

GameMenuSwitch

SpriteStrings

Klasse diagram nieuw systeem



Entity - Wordt geïdentificeerd door een ID dit staat het toe deze later gemakkelijk op te halen. Houd een lijst bij van alle Components die een entiteit heeft.

Trustee - Basis klassen van het bestaande systeem, houd bij hoeveel geld een bedrijf heeft. Wordt verder gebruikt om integratie met het bestaande systeem gemakkelijker te maken.

Market - Koopt en verkoop grondstoffen, bepaalt de gevraagde prijs aan de hand van de basis prijs van een grondstof.

EntityManager - Maakt het mogelijk Entiteiten op te zoeken aan de hand van naam, type en ID.

Resource - Opslaan gegevens van grondstoffen.

EntityComponent - Klasse waarvan alle components overerven. Hierin staan methodes voor Init en Execute.

CompanyController - Beheert alle acties die een entiteit kan ondernemen en voert deze in de juiste volgorde uit.

Action - Een actie die een entiteit kan ondernemen. Houd een lijst van conditions bij waaraan de action moet voldoen om uitgevoerd te kunnen worden. Heeft een of meer components die nodig zijn om de action uit te kunnen voeren.

Condition - Gevens over een conditie, welke variabele welke waarde moet hebben.

Operator - Opslaan en uitvoeren van de operation die de conditie moet doen.

Bijlage IV: User Requirement Specification

In dit document worden de eisen en wensen van de opdrachtgever uitgewerkt.

MoSCoW

Nieuw bedrijven toevoegen	M
Nieuw grondstoffen toevoegen	M
Nieuw betaalmiddel toevoegen	C
Overzicht bedrijven	M
Overzicht markten	C
Bedrijven overzicht sorteren doormiddel van filters (Kostprijs, inkoop/verkoop)	S
Aanpassen waardes bedrijven	M
Weergeven bedrijven binnen een specifieke markt	W
Totaal overzicht van een Markt(totale grondstof productie/consumptie, aantal bedrijven)	W
Aanpassen welke type betaling een bedrijf accepteert	S
Aanpassen onder welke voorwaarden een bedrijf betalingsmiddelen(krediet, valuta) accepteert	M
Acties die en bedrijf kan ondernemen aanpassen	M
Upgrades voor bedrijven aan/uit zetten	S
Upgrades aanpassen(voorwaardes, kosten, effect)	C
Nieuwe grondstoffen automatisch weergeven in markt visualisatie.	M
Weergeven welke acties een bedrijf kan ondernemen(UI in game)	M

Legenda		
M	=	Must have
S	=	Should have
C	=	Could have
W	=	Will not have

Niet functionele eisen

De niet functionele eisen zijn opgesteld aan de hand van

- Het project/systeem moet overzichtelijker worden.
- Gemakkelijk nieuw gedrag toevoegen.
- Aanpassingen moeten met zo min mogelijk programmeren te realiseren zijn
- Bestaand gedrag zo veel mogelijk hergebruikt worden

De niet functionele eisen zijn gebaseerd op de lessen die geleerd zijn met het ontwikkelen van de fysieke versie. Hierbij gaat het vooral om aspecten van paperprototyping.

Acties

Acties	Omschrijving
Koop/Verkoop	<p>Kopen/Verkopen van grondstoffen aan een of meer andere bedrijven.</p> <p>Bedrijven sluiten een “TradeContract” af met alle bedrijven waaraan zij grondstoffen kopen/verkopen.</p> <p>Bedrijven zullen altijd eerst proberen grondstoffen te kopen/verkopen aan bedrijven waarmee zij een bestaand “TradeContract” hebben. Als een bedrijf geen bestaande “TradeContract” heeft (of deze al verzadigd zijn) wordt er een nieuw “TradeContract” aangegaan, als dit mogelijk is.</p> <p>Als de productie van bedrijven veranderd (door upgrades of andere oorzaken) worden waar mogelijk bestaande “TradeContract” aangepast anders wordt er (als het mogelijk is) een nieuwe “TradeContract” aangegaan. In het geval van een verlaging van de productie worden “TradeContract” stopgezet.</p>
Converteer	<p>Converteren (produceren) van een of meer grondstoffen naar een andere grondstof.</p> <p>Het produceren van grondstoffen kost Labour in de vorm van personeel. Hierdoor kost deze actie een bedrijf geld (loon).</p> <p>Als het bedrijf niet voldoende grondstoffen heeft kunnen inkopen worden alle beschikbare grondstoffen omgezet. De exacte hoeveelheid wordt bepaald aan de hand van hoeveel procent van de grondstoffen wel ingekocht is. In het geval van meerdere benodigde grondstoffen wordt gebruik gemaakt van het laagste percentage.</p>
Overbid	<p>Meer bieden als de huidige markt waarde van een grondstof om zo voorrang te krijgen met het inkopen van deze grondstof.</p> <p>Het bedrijf kan ervoor kiezen een bestaand “TradeContract” voor een specifieke grondstof te overbieden om zo zelf een nieuw “TradeContract” met het betreffende bedrijf af te sluiten. Hierdoor wordt het bestaande “TradeContract” stopgezet.</p> <p>De waarde waarmee overbieden kan worden (en moet worden) kan per bedrijf en bedrijfstype verschillen. Het gaat hierbij om een percentage van de totale prijs van het bestaande “TradeContract”. Deze prijs moet het bedrijf daarna blijven betalen (zolang als het “TradeContract” blijft bestaan).</p>
LoanMoney	<p>Een x bedrag lenen van een financiële instelling.</p> <p>Aangaan van een lening om kosten van het bedrijf te kunnen betalen. Het te lenen (x) bedrag wordt bepaald aan de hand van het benodigde bedrag en de maximale hoogte van de maand afbetaling (die het bedrijf zich kan veroorloven).</p> <p>Een lening kan alleen worden afgesloten bij een bank die in het zelfde dorp gevestigd is als het bedrijf.</p> <p>Een bedrijf kan een lening aangaan vanwege de volgende 2 redenen.</p>

	Opstart kosten van een nieuw bedrijf of upgrade kosten van een bestaand bedrijf te kunnen financieren. Deze kosten bestaan uit het aankopen van bouwmaterialen en personeelskosten. Voor een nieuw bedrijf komen hierbij ook nog de benodigde operatie kosten van de eerste maand bij.
Upgrade	<p>Het bedrijf upgraden om zo de productie te verhogen.</p> <p>Tijdens het upgraden kan het bedrijf geen Converter actie ondernemen hierdoor is er geen inkomen. Voor het upgraden van een bedrijf wordt het personeel (Labour) gebruikt hierdoor heeft het bedrijf wel nog kosten.</p> <p>Voor de upgrade moet het bedrijf van tevoren grondstoffen aanschaffen om de betreffende upgrade te kunnen gebruiken. Hiervoor kan het bedrijf ook zelf geproduceerde grondstoffen gebruiken.</p> <p>Wood, Tools en Labour worden normaal gesproken gebruikt voor upgrades. Na een upgrade wordt de grondstof input, output en storage aangepast. Verder krijgt het bedrijf ook een nieuw uiterlijk.</p>
Investeren	<p>Investeren in een nieuw op te richten bedrijf binnen het dorp. Als voorwaarde hiervoor moet er een lege "Tile" zijn binnen het dorp en moet er overschot of tekort zijn aan grondstoffen.</p> <p>Het type bedrijf wordt bepaald aan de hand van welke grondstoffen er een tekort (of juist overschot) is. Als er van meerdere grondstoffen een tekort (of overschot is) is wordt er gekozen voor de grondstof met de hoogste marktwaarde en naar welke productie grondstof de hoogste geschatte winst oplevert.</p> <p>Al Banken verstrekken altijd een lening nieuw opgestart bedrijf, spelers kunnen ervoor kiezen dit te weigeren.</p>

Beperkingen

Beperkingen	
Affordable	Uitgeven van geld kan alleen gedaan worden als een bedrijf deze kosten kan veroorloven.
CanStore	Inkoop en converteren van grondstoffen kan alleen ondernomen worden als het bedrijf de benodigde opslag ruimte heeft voor de grondstoffen.
CanSell	Converteer actie kan alleen ondernomen worden als het bedrijf een koper heeft (of kan vinden) voor de gemaakt grondstoffen.
IsMakingProfit	Een bedrijf kan alleen uitgebreid worden als het momenteel winst maakt.
CanDoLoanPayment	Een lening mag alleen afgesloten worden als het bedrijf genoeg winst maakt om de maandelijkse betaling te kunnen betalen.

Regels

De spelregels zijn opgenomen als toekomstige mogelijk uitbreiding van het systeem om naast Regels, Acties en Beperkingen van entiteiten ook spel regels aan te kunnen passen.

Hiervan zijn voorbeelden opgenomen van zowel het fysieke als het digitale spel om verschillende mogelijkheden (voor de zelfde spelregels) weer te geven.

Beurt

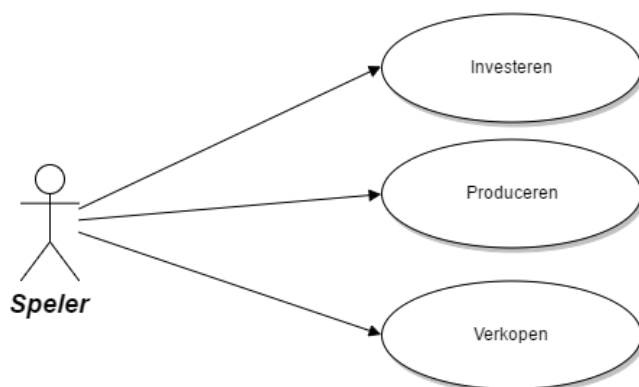
Bordspel

Speler krijgt de mogelijkheid 1 of meer nieuwe bedrijven te kopen(in te investeren)

Speler kan al zijn bedrijven 1 keer laten produceren(volgorde mag zelf bepaald worden). De output van een bedrijf kan gebruikt worden als input bij een ander bedrijf. output kan verkocht worden of bewaard worden tot een volgende beurt.

Speler kan de kosten voor inkopen grondstoffen en gebouwen betalen met zowel krediet of goud. Als betaling voor het verkopen krijgt de speler altijd goud.

Use Cases



Investeren	
Actor(s)	Speler
Pre-conditions	Er is tenminste 1 open investeringskaart
Normal flow	1. Systeem bepaald start bod

	2. Speler(s) bieden met krediet op investering 3. Speler met het hoogste bod plaatst de geboden hoeveelheid krediet op het bord 4. Systeem verwijderd de kostprijs(van het gebouw) uit de markt
Post conditions	Speler heeft het gebouw in bezit
Alternate flows	Geen andere Speler bied op het betreffende gebouw 1. Speler koopt het gebouw voor het start bod

Produceren	
Actor(s)	Speler
Pre-conditions	Speler heeft een of meer bedrijven, Speler heeft geld/krediet
Normal flow	1. Systeem bepaald inkoop prijs 2. Speler gebruikt grondstoffen voor productie 3. Systeem zet input grondstoffen om naar output grondstoffen 4. Systeem vernietigd input grondstoffen
Post conditions	Speler krijgt output grondstoffen in zijn inventaris
Alternate flows	Speler gebruikt zijn eigen grondstoffen(als deze aanwezig zijn) 1. Begin bij stap 2

Verkopen	
Actor(s)	Speler
Pre-conditions	Speler heeft een of meer grondstoffen
Normal flow	1. Systeem bepaald de markt waarde voor de grondstoffen. 2. Speler bepaald betaalmiddel 3. Speler ontvangt markt waarde in zijn gekozen betaalmiddel
Post conditions	
Alternate flows	Als het betaalmiddel goud is en er niet genoeg aanwezig is wordt er een Credit Call uitgevoerd voordat de Speler betaald wordt.

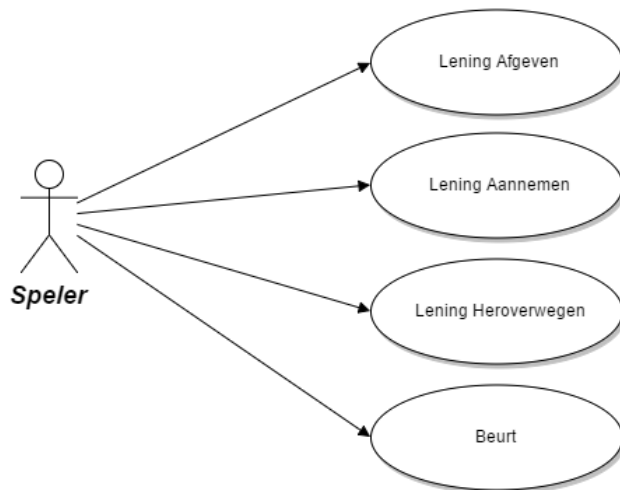
Digitale spel

Speler investeert in gebouwen(geeft een lening) spel simuleert voor het betreffende bedrijf de komende maand aan productie/kosten, hierna wordt bepaald of het gebouw de investering(bedrag per maand voor x maanden) kan betalen of niet. Zo ja krijgt de speler geld, zo niet moet de speler zijn lening heroverwegen(gebouw overnemen, of aantal maanden terugbetalen verhogen).

De speler kan ook geld lenen van succesvolle bedrijven, de speler kan daarna dit geld gebruiken om meerdere leningen aan bedrijven te geven.

Simuleren gebeurt nadat de speler op een continue knop geklikt heeft.

Use Cases



Lening afgeven	
Actor(s)	Speler
Pre-conditions	Er is een bedrijf die een lening wilt/nodig heeft
Normal flow	<ol style="list-style-type: none"> Speler bepaald terugbetaal tijd Speler bepaald hoogte lening Systeem bepaald maand bedrag Speler en bedrijf accepteren bedragen. Bedrijf ontvangt bedrag in Goud
Post conditions	
Alternate flows	Maand bedrag ligt te hoog(of te laag) bedrijf wijst lening af.

Lening aannemen	
Actor(s)	Speler
Pre-conditions	Er is een bedrijf met een geldoverschot.
Normal flow	<ol style="list-style-type: none"> Bedrijf bepaald terugbetaal tijd Bedrijf bepaald hoogte lening Bedrijf bepaald maand bedrag Speler accepteert lening
Post conditions	
Alternate flows	

Lening heroverwegen	
Actor(s)	Speler
Pre-conditions	Lening kan niet afbetaald worden
Normal flow	1.
Post conditions	
Alternate flows	Als de Speler een van zijn leningen niet kan af betalen gaat de Speler failliet(Game over)

Beurt	
Actor(s)	Speler
Pre-conditions	Speler heeft alle andere acties uitgevoerd(of wilt/kan ze niet uitvoeren)
Normal flow	<ol style="list-style-type: none"> 1. Speler selecteert mogelijkheid om naar de volgende beurt te gaan. 2. Systeem simuleert de volgende beurt(1 maand).
Post conditions	
Alternate flows	

Beter, is nu niet duidelijk

State diagram toevoegen

[Afbetalen Krediet](#)

Bordspel

Voor een “Credit Call” wordt een dobbelsteen gegooid, doormiddel van deze dobbelsteen wordt bepaald welke gebieden in de stad terugbetaald moeten worden. Iedere speler krijgt de mogelijkheid om zijn/haar krediet af te betalen(alles wat betaald kan worden moet ook betaald worden).

Spelers die niet genoeg goud hebben om dit te kunnen betalen moeten met andere spelers onderhandelen over een lening. De andere spelers kunnen zelf bepalen of zij die wel of niet willen.

Mocht het voor de speler niet mogelijk zijn alles af te betalen zakt hij/zij een “trust level” en wordt er nog een “Credit Call” gehouden(dus het is ook in het belang van de andere spelers om dit te voorkomen).

Use Cases

Credit Call	
Actor(s)	Speler
Pre-conditions	Speler heeft krediet op het bord liggen
Normal flow	<ol style="list-style-type: none"> 1. Systeem bepaald welk krediet afbetaald moet worden. 2. Speler betaald zijn schulden af 3. Systeem bepaald of de Speler stijgt in trust level
Post conditions	
Alternate flows	Als de speler zijn schulden niet kan afbetalen ga naar [betalings probleem].

Betalingsprobleem	
Actor(s)	Speler
Pre-conditions	
Normal flow	<ol style="list-style-type: none"> 1. Systeem bepaald het betalingsprobleem van de Speler 2. Speler onderhandeld met andere Spelers om zijn betalingsprobleem op te lossen. 3. Speler betaalt(met geleend geld) het betalingsprobleem
Post conditions	
Alternate flows	Speler kan nog steeds zijn schulden niet afbetalen ga naar [Degradieren]

Promoveren	
Actor(s)	Speler
Pre-conditions	Speler zit niet in trust level AAA,

Normal flow	1. Speler stijgt een trust level 2. Speler verdeelt zijn krediet over de nieuwe ring
Post conditions	
Alternate flows	

Degraderen	
Actor(s)	Speler
Pre-conditions	
Normal flow	1. Speler daalt 1 trust level 2. Systeem gaat naar [Credit call]
Post conditions	
Alternate flows	Speler zit op het moment van een betalingsprobleem in het zwarte gat de Speler gaat hierdoor failliet(game over).

Digitale spel

Na iedere maand moeten entiteiten binnen het spel een deel van hun schulden terug betalen(dit geldt ook voor de speler). De hoeveelheid die terug betaald moet worden wordt bepaald aan de hand van het totale bedrag van de lening en de looptijd hiervan.

UI

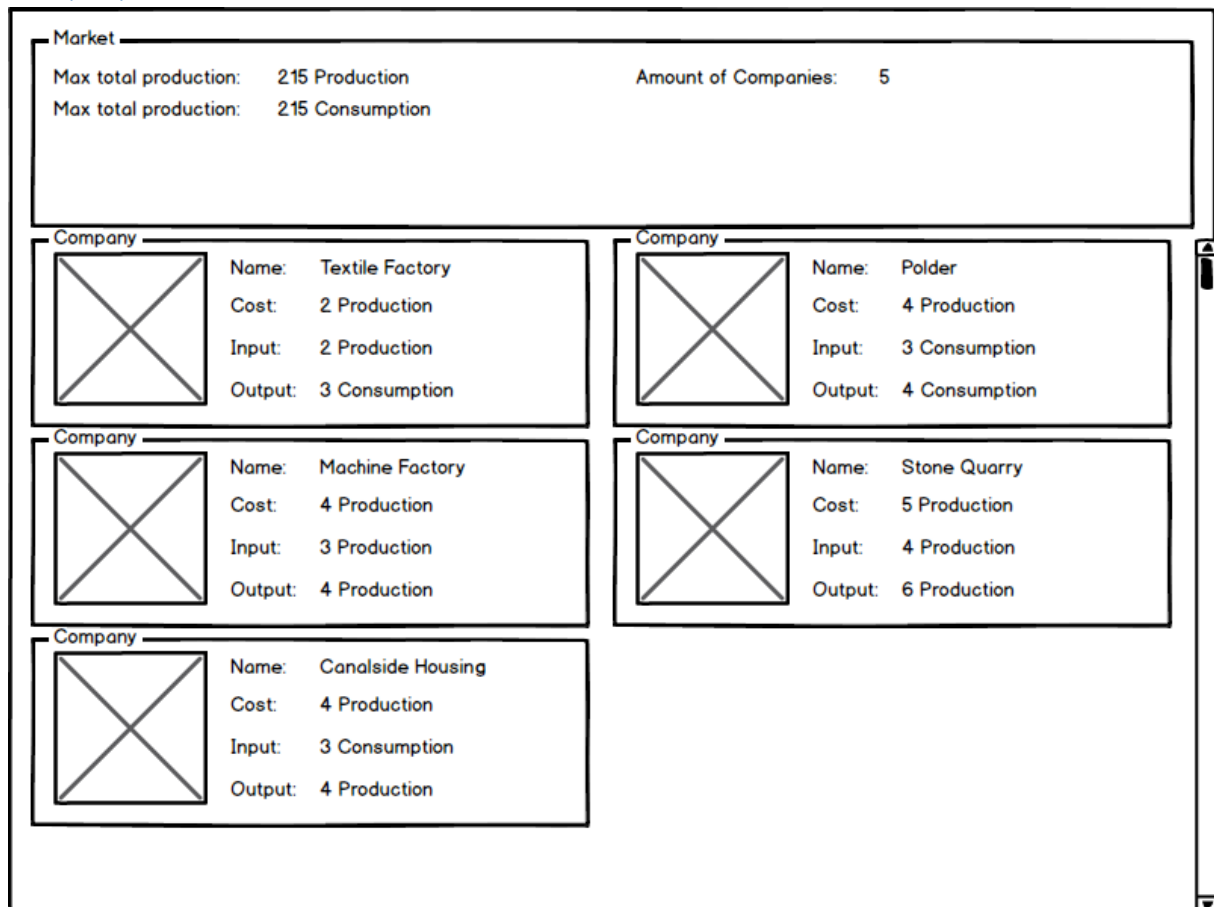
De hieronder staande views worden gerealiseerd doormiddel van Editor Windows binnen Unity3D.

Diagram View

Overzicht van het huidige diagram. Dit moet gekoppeld worden aan welke prefab/gameobject in Unity3D geselecteerd is(als ondersteuning van de bedrijven). Mogelijk in dit overzicht ook de mogelijkheid bieden om het betreffende diagram aan te passen.

Machinations diagram of Micro Machinations diagram weergeven(afhankelijk van wat mogelijk is).

Company Overview



Overzicht van alle type bedrijven binnen het spel, van ieder type bedrijf kunnen er meerdere instanties aanwezig zijn binnen de spelwereld. Hiervoor is gekozen omdat de variabelen en acties van het type bedrijf afhankelijk zijn. Hierop is binnen het bordspel wel 1 uitzondering maar dit valt voor het onderzoek buiten de scope.

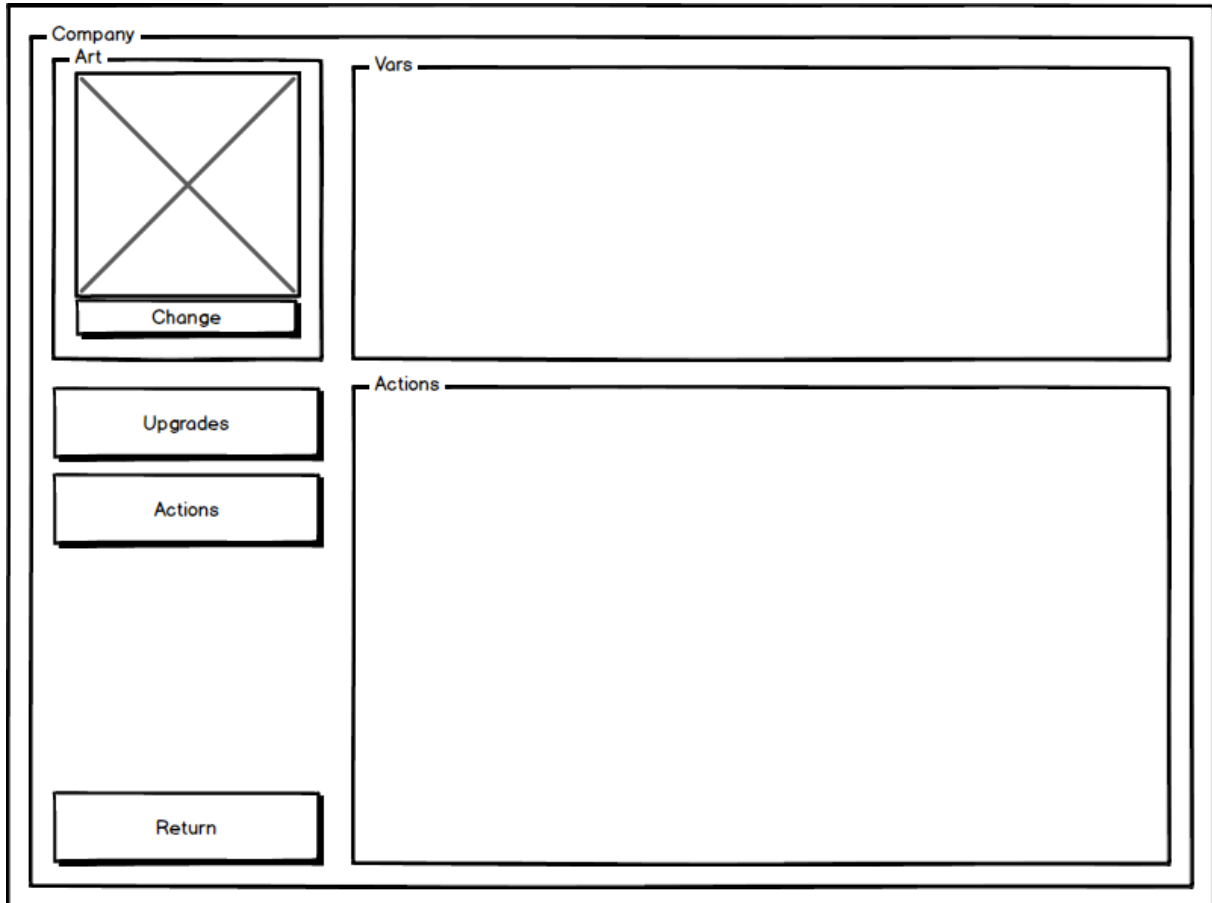
De dingen die hierin voor ieder bedrijf zichtbaar zijn komen uit het bordspel en kunnen voor het digitale spel veranderen.

In dit scherm is basis informatie over alle bedrijven te vinden. In het onderdeel Market kan informatie weergegeven worden over de totale productie, begin prijs, hoeveelheid verschillende bedrijven en andere benodigde informatie(exacte informatie kan op een later moment bepaald worden aan de hand van de eisen en wensen van de opdrachtgever).

In dit overzicht op een bedrijf klikken zorgt er voor dat het volgende scherm(CompanyView) geopend wordt.

Doormiddel van Filters kunnen de bedrijven waar nodig gesorteerd worden.

Company View

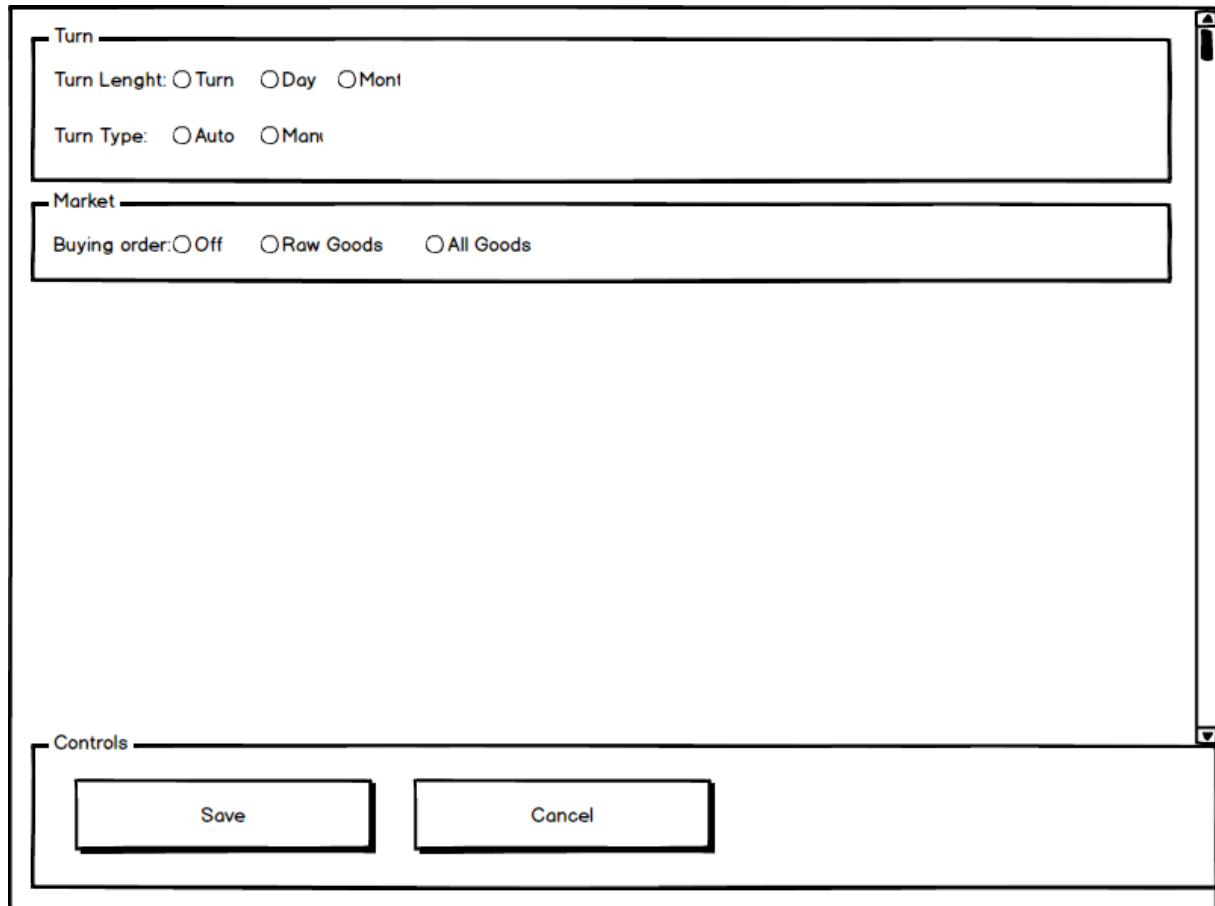


The diagram illustrates the layout of the 'Company View' form. It is enclosed in a double-line border. On the left side, there is a vertical stack of elements: a 'Company Art' section containing a square placeholder with a diagonal 'X' and a 'Change' button below it; an 'Upgrades' button; an 'Actions' button; and a 'Return' button at the bottom. On the right side, there are two large rectangular input areas. The top one is labeled 'Vars' and the bottom one is labeled 'Actions'.

In dit scherm kunnen de basis gegevens van het bedrijf aangepast worden doormiddel van de input velden. Actions en Upgrades kunnen zichtbaar gemaakt worden doormiddel van de daarvoor betreffende knop. Hierna moet in het onderste scherm nu genoemd "actions" de mogelijkheid geboden worden deze aan te passen.

De precieze implementatie hangt af van de gekozen techniek.

Rules Overview



The image shows a 'Rules Overview' dialog box with a scrollable content area. The dialog has a title bar and a standard window border. Inside, there are three sections: 'Turn', 'Market', and 'Controls'. The 'Turn' section contains 'Turn Lenght' (sic) with radio buttons for 'Turn', 'Day', and 'Month', and 'Turn Type' with radio buttons for 'Auto' and 'Manual'. The 'Market' section contains 'Buying order' with radio buttons for 'Off', 'Raw Goods', and 'All Goods'. The 'Controls' section at the bottom contains two buttons: 'Save' and 'Cancel'. A vertical scrollbar is visible on the right side of the dialog.

Turn

Turn Lenght: ☐ Turn ☐ Day ☐ Month

Turn Type: ☐ Auto ☐ Manual

Market

Buying order: ☐ Off ☐ Raw Goods ☐ All Goods

Controls

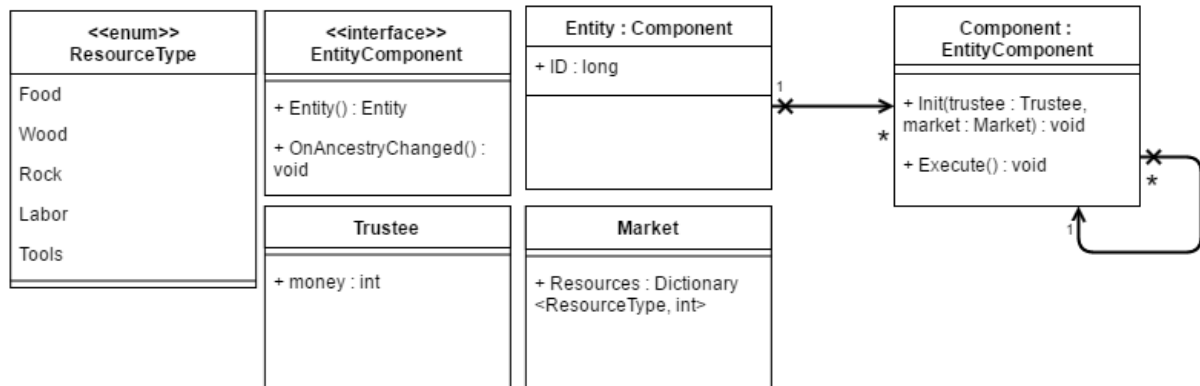
Save Cancel

Weergeven van alle regels waarin de mogelijkheid bestaan regels aan of uit te zetten. Regels uitbreidbaar maken aan de hand van de Micro Machinations diagrammen.

Bijlage V: Klasse diagram POC ECS

Ontwerp Proof of concept Entity Component System

Klasse diagram



Beschrijving

Entity - Wordt geïdentificeerd door een ID dit staat het toe deze later gemakkelijk op te halen. Houd een lijst bij van alle Components die een entiteit heeft.

EntityComponent - Standaard methodes die door iedere Component gebruikt kunnen worden. Hierin zijn een Condition, Action en Init methode aanwezig.

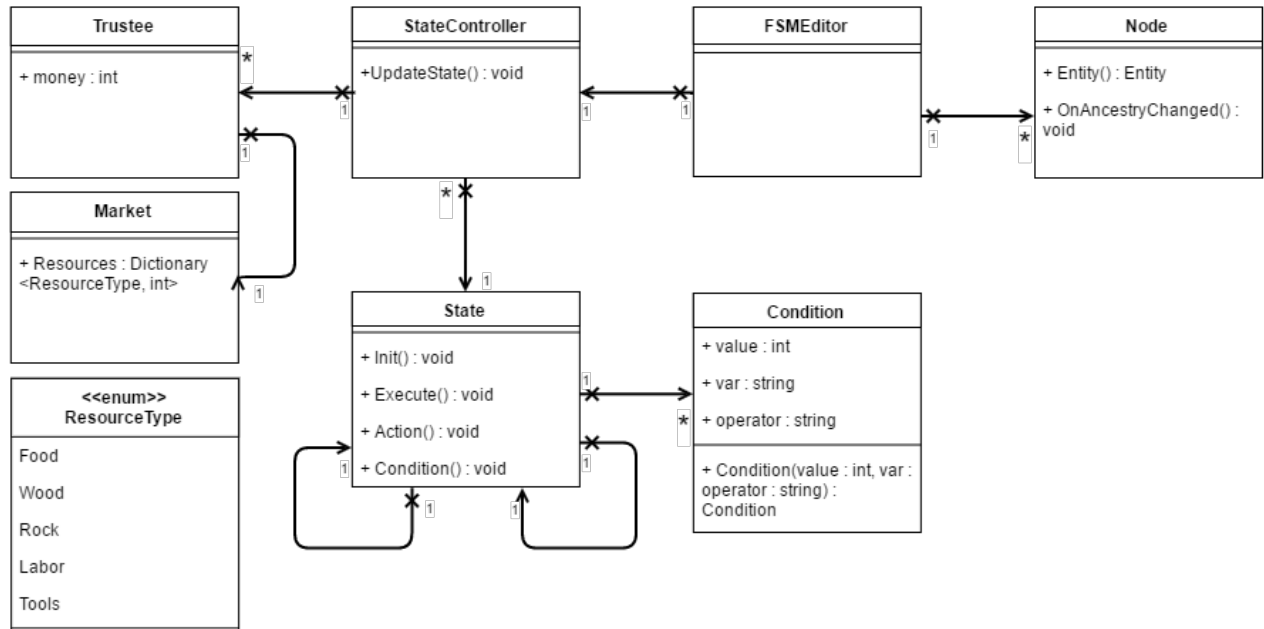
Component - Component overerft van EntityComponent en definieert de inhoud van de methodes.

ResourceType - Enum met de verschillende grondstoffen die in het spel gebruikt kunnen worden.

Bijlage VI: Klasse diagram POC FSM

Ontwerp Proof of concept Finite State Machine

Klasse diagram



Beschrijving

Trustee - Houd de hoeveelheid geld/krediet van een entiteit bij.

Market - Opslaan van grondstoffen. Wordt door entiteiten gebruikt om grondstoffen onderling te verhandelen.

ResourceType - Enum met de verschillende grondstoffen die in het spel gebruikt kunnen worden.

StateController - Lijst van alle mogelijk states. Verantwoordelijk voor het bijhouden en uitvoeren van de actieve state.

State - Lijst van alle mogelijke condities voor de specifieke state. Houd mogelijk transistions naar anders state bij. Actie wordt hierin geprogrammeerd.

FSMEditor - Editor Window verantwoordelijk voor het visueel weergeven van Nodes en het omzetten hiervan naar States en de StateController.

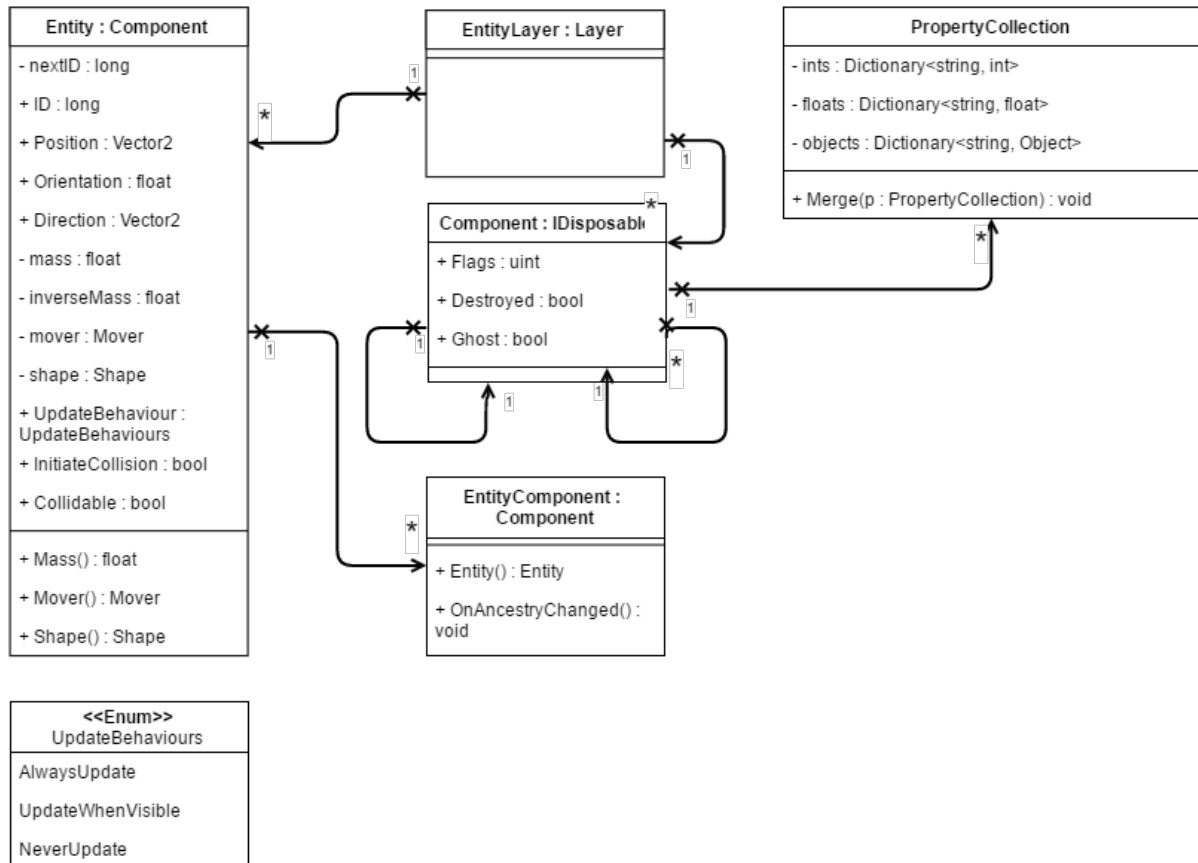
Node - Opslaan gegevens van visuele weergaven van states. Het gaat hierbij om transiitons, id, type, name en conditions.

Condition - Bevat gegevens van een condition. Het gaat hierbij om varaibele, value en operator.

Bijlage VII: Klasse diagram POC Adapt Tower

Ontwerp Proof of concept Adapt Tower

Klasse diagram



Beschrijving

Entity - Wordt geïdentificeerd doormiddel van een ID dit kan later gebruikt worden om specifieke Entiteiten te vinden. Heeft een aantal standaard variabelen en components om gedrag te creëren dat iedere in game entiteit moet kunnen hebben. Denk hierbij aan CollisionDetection en Movement.

Component - Compoent definieert de basis (base) functionaliteit die alle components kunnen overerven. Een component moet altijd een onderdeel zijn van een GameObject.

EntityComponent - Overerft van Component. Een EntityComponent moet altijd onderdeel zijn van een Entity.

EntityLayer - Koppeling tussen components en entiteiten. Heeft functionaliteit om components acties te laten uitvoeren met betrekking tot Entiteiten.

PropertyCollection - Data storage klasse die door Components gebruikt kan worden om variabelen tussen verschillende Components te delen. Dit zorgt ervoor dat Components niet naar andere Components (voor de variabelen) hoeven te verwijzen.

UpdateBehaviours - Enum die gebruikt wordt om te bepalen hoe vaak en wanneer een Entity geupdate moet worden.