SYSTEMATISK SPILANALYSE

En grundlæggende guide

Mikkel Lodahl og Kenneth Lodahl Nielsen

ERHVERVSAKADEMI DANIA

INSTITUT FOR DANSK SPILUDVIKLING

LIONFISH CROSSMEDIA

Systematisk spilanalyse – en grundlæggende guide

Bogen er sat med Adobe Garamond og Minion

Forsideillustration: foto af Thomas Leuthard (www.thomasleuthard.com) fra Albumarium.com

Udgivet af Institut for Dansk Spiludvikling, Lionfish Crossmedia og Erhvervsakademi Dania

ISBN 978-87-997705-7-1 (epub)

ISBN 978-87-997705-8-8 (Kindle)

ISBN 978-87-997705-9-5 (pdf)

2. udgave, 1. oplag 2015

INDHOLD

INDLEDNING

Det følgende er et forsøg på at lave en samlet oversigt over forskellige fagligt systematiske måder at analysere inden man begynder at udvikle et spil.

Spilbranchen er præget af en stærk tradition for at være selvlært, og mange spiludviklere har traditionelt set ingen formel softwareudviklingsuddannelse bag sig. Det er ved at ændre sig i takt med at uddannelsesinstitutioner i højere og højere grad specialiserer sig i eller giver plads til udvikling af computerspil.

Udfordringen er imidlertid, at når den hårdt erhvervede teori fra uddannelserne i spiludvikling møder en praksis med en selvlært tradition, bliver det svært at gøre sig det helt klart, hvordan man får mest muligt ud af sin uddannelse. Det håber vi fra Institut for Dansk Spiludvikling at kunne være med til at afhjælpe.

Selv om Erhvervsakademi Dania i Grenå hvor denne bog stammer fra fortsat huser den eneste videregående uddannelse i Danmark kun med fokus på spil, er udfordringen nemlig delt blandt de fleste kandidater, der har spilrettede elementer i deres baggrund. Samtidig er stort set alle klassiske softwareudviklingsuddannelser også rugesteder for spilinteresserede programmører og designere, der ofte kan have god brug for en hjælpende hånd til at få mest muligt ud af de uddannelser, de har taget.

Bogen her er det første skud på stammen i hvad vi har planlagt som et trekløver, der skal dække de klassiske softwareudviklingsfaser – analyse, design og implementering.

I denne bog beskæftiger vi os altså med analyse-fasen, som vi har inddelt i tre afdelinger.

Content Analysen dækker over hvordan man analyserer og taler om computerspil som medie.

Business Analysen dækker over hvordan man kæder sin spilidé sammen med en realistisk vurdering af, om der kan skabes en forretning ud fra den.

Den Objektorienterede Analyse dækker over selve planlægningen af systemets arkitektur og dynamik som software. Dette er den disciplin, der lægger sig mest op af klassisk datalogi, og selv om den er placeret sidst, kan spilinteresserede dataloger måske med fordel starte der.

Endelig slutter vi bogen af med et lille afsnit, der kommer mere detaljeret ind på sammenhængen imellem de tre analyseområder. I praksis gentages analyserne ofte flere gange i en udviklingsproces.

Bogen er resultatet af fem års praksis på Erhvervsakademi Dania i Grenå, hvor modellen gradvist har udviklet sig og er blevet afprøvet sammen med studerende og spilvirksomheden Lionfish Crossmedia. Forfatterne er sig denne gæld meget bevidst og takker de tålmodige studerende og udviklere, der har lagt undervisning og opgaver til og arbejde i udviklingen.

Specielt går vores tak til de studerende, der har været deltagere i fokusgrupper, der løbende er kommet med input til tekstens udformning. Det drejer sig om de datamatikerstuderende Frederik Kjær Svendsen, Trine Strøm Harder Bang, Mark Petersen, Dion Sune Jensen, Kenneth Andersen Glyngfeldt, Jonas Bjørn Pedersen, Morten Hyldgaard og Daniel Boendorf Lauridsen – der alle skal have tak!

God læselyst!

Kenneth Lodahl Nielsen og Mikkel Lodahl,

(keln@eadania.dk og milo@eadania.dk)

Grenaa, juni 2015

CONTENT ANALYSE

Systematik >< Fornemmelse

Potter Stewart, der var højesteretsdommer i USA, dømte i 1964 i en sag, hvor han skulle definerer, hvornår noget var pornografi. Han vendte og drejede en lang række definitioner, men i sidste ende blev han enig med sig selv om, at hans assistent Alan Novak havde ramt hovedet på sømmet med beskrivelsen: "I know it, when I see it."

Som computerspiludvikler er man ofte i samme situation. Hvis man skal beskrive, hvad det er, der faktisk karakteriserer et godt eller et sjovt computerspil, så har vi svært ved at formulere os på en måde, der siger noget rammende og brugbart. Det virker på mange måder nemmere at pege på ting, vi kan se fungere som eksempler på det. For eksempel kan vi synes, at Bioshock er et fantastisk spil, men have svært ved at sige andet end "Lad os lave et spil ligesom Bioshock!" og så kaste os ud i det.

Desværre er det kun en udvej, der kan bruges, hvis man ikke rent faktisk skal bruge sin indsigt til noget. Som fan, jurist eller kritiker er det til nød godt nok at kunne genkende et godt spil, når man ser det. Som spiludvikler skal man imidlertid kunne se et godt spil og forstå hvad det er, der gør det godt i en sådan grad, at man kan tage inspiration fra det i sit videre arbejde. Man skal kunne spille et spil og pege ikke på spillet, men på en del af spillet som noget, man vil opnå, undgå, viderebygge, undersøge i sit eget virke og værk. Man skal kunne se på Bioshock og sige, "Det jeg synes, virker godt er, hvordan historien fortælles primært igennem miljøet. Lad os designe levels, der også fortæller en historie gennem deres tilstand."

Man skal bruge en systematisk måde at tale om spillene på.

Men kan man ikke bare lave en prototype og så se, om dét er et godt spil? I princippet jo, men der er mindst tre gode argumenter imod den tilgang:

- *Prototyper er ikke dækkende for alle typer spil.* Specielt i AAA-industrien er det næsten umuligt at lave prototyper, der rent faktisk illustrerer det færdige spil, fordi så meget af interaktionen med spillet ligger i assets af den ene eller den anden art. En bestemt kampmekanik i et actionspil virker måske sjov som prototype, men når først de forskellige, high-end grafiske teksturer og animationer er kommet på, virker den i stedet langsom og klodset. Hvad værre er: måske er der ting, der ikke prototyper godt, der alligevel kan fungere, når de er implementeret i kontekst.
- *Vi tænker, før vi handler, specielt i en langvarig proces.* Selv om der er øjeblikke i enhver kreativ proces, hvor det føles som om, værket tager over og man ikke tænker over, hvad

man laver, så er der lange perioder, hvor man tænker sig frem mere end man føler sig frem. I en lang og kompleks proces som spiludvikling vil man naturligt have brug for at stoppe op og forsøge at danne sig et overblik. Hvad enten vi kan lide det eller ej, har vi brug for en systematisk måde at gøre disse overvejelser på for at komme nemmere ind og ud af arbejdet.

- *Vi arbejder ofte sammen*. Når vi udvikler computerspil, gør vi det næsten altid i teams. Hvis jeg viser dig en bane i Super Mario Bros. 3 og siger: "Den er fed! Den laver vi!" hvad vil du så gøre? Kopiere den pixel for pixel? Finde ud af, hvad det er, der får den til at fungere og forsøge at kopiere det? Hvad hvis vi laver et skydespil? Hvordan vil du tage det, jeg synes er fedt i Mario-banen og tage det over i en anden genre? Det kan du kun, hvis du har en eller anden systematisk måde at sortere og diskutere din og min fornemmelse af, hvad der fungerer i spillet.

De tre argumenter fremhæver, at vi har brug for noget, der kan identificere bestemte dele af spil, kan bruges til at systematisere praktiske overvejelser i spiludvikling og kan bruges på tværs af forskellige spilgenrer.

MDA-frameworket

Ét bredt benyttet bud på en systematisk måde at tale om spils indretning på, er MDA-frameworket. Det er en forståelse af, hvordan computerspil bredt fungerer, der er udviklet af Robin Hunicke, Mark Leblanc og Robert Zubek. Modellen er udviklet via workshops på Game Developers' Conference, der er computerspilbranchens centrale samlingspunkt, og de tre herrer slog samtidig deres folder som forskere på de tekniske universiteter Northwestern University, Illinois, og MIT.

Modellen afspejler dette ved både at være forankret i den gængse, praktiske arbejdsmetode blandt spiludviklere og i mere akademisk softwareudviklingsmetode.

Kernen i modellen er forholdet imellem tre store, strukturelle dele af computerspillet som fænomen – Mechanics, Dynamics og Aesthetics.

Mechanics er en samlebetegnelse for de dele af et spil, som spiludviklerne har direkte indflydelse på. Kode, regler, assets, platform, styring, historie og så videre. Det er udformningen og udvælgelsen af legoklodserne.

Dynamics er en samlebetegnelse for al den interaktion, spilleren kan have med mechanics. Lad os sige, at som mechanics laver vi en regel, der siger, at hvis man rammer en koopa forfra i Mario, tager man skade, men hvis man rammer den ovenfra, slår man den ind i skjoldet. Skjoldet kan man skubbe til og rammer det andre koopaer forfra, dør de. Jo flere det rammer, jo flere points får man. Med nok points får man et ekstra liv. Dynamics er så det, at spilleren enten kan hoppe på en enkelt Koopa og få de point, eller bruge skjoldet til at få mange flere points. Dynamics

behøver ikke at være aktivt fra spillerens side, men kan også betegne modtagelsen af informationer gennem passive dele af spillet, for eksempel cutscenes. Hvis mechanics er udformningen og udvælgelsen af legoklodserne, så er dynamics udforskningen af forskellige måder at sætte dem sammen på.

Aesthetics er en samlebetegnelse for den oplevelse, spilleren får ud af dynamisk at interagere med mechanics. Lad os sige, at vi i mechanics laver reglerne for kamp med koopaen som nævnt oven for, og vi i dynamics hopper på den første koopa i en række af koopaer, hvorefter vi skubber til skjoldet. Skjoldet stryger hen over skærmen, dræber de andre koopaer, og når det rammer den sidste har vi fået points nok til at få et ekstra liv. I mechanics har vi placeret det sådan, at der dukker tal op, der tæller de stigende points og at når der kommer et ekstra liv lyder en positiv og glad lydeffekt. Kombinationen af behændighed, fart, points og ekstra liv giver spilleren en oplevelse af succes og glæde. Denne følelse er aesthetics, eller oplevelsen af succesfuld dynamisk interaktion med mekanikkerne. Hvis mechanics er udformningen og udvælgelsen af legoklodserne og dynamics er udforskningen af forskellige måder at sætte dem sammen på, så er aesthetics oplevelsen af at se på den figur, man bygger af klodserne.

MDA-frameworket opfylder på mange måder vores ønsker til en måde at tale om computerspil på. Der identificeres forskellige dele af spillet, det bliver klart hvilke dele man kan tage hånd om som udvikler.

Samtidig giver frameworket en forståelse for forskellen mellem at være udvikler og være spiller. Hvor udvikleren har direkte adgang til mechanics og forsøger at styre de mulige dynamics i håbet om at skabe en bestemt aesthetic-oplevelse, ser det omvendt ud for spilleren. Spilleren har en aesthetic-oplevelse, gennem hvilken hun eller han kan udføre dynamics i et forsøg på fuldstændig at afdække de mechanics, der er.

Endelig svarer de tre dele godt til klassisk forståelse af softwareudvikling, hvor man opbygger en statisk model, som brugerne kan interagere med dynamisk, for at opnå et bestemt resultat – i spil er det bare ikke et bestemt resultat, men en bestemt oplevelse.

MDA, iterationer og tests

Men bare fordi man har en systematisk måde at tale om og udvikle et spil, betyder det ikke, at man fra starten af kan regne ud, hvordan man skal lave et godt spil. Som spiludvikler kan man klarlægge, hvad der er mechanics, og hvilke dynamics, de mechanics formodentlig kaster af sig. Men man har kun indirekte kontrol over aesthetics og kan nok næppe kortlægge dynamics fuldstændigt.

Det er med andre ord en god idé stadig at tænke prototyper ind i udviklingen. Typisk vil man gøre dette ved hjælp af iterationer. En iteration betyder et gennemløb, hvilket vil sige, at man laver et

spil, der inkorporerer mechanics, dynamics og aesthetics, men som kører og kan afprøves. Det vil sige, at en iteration skaber en prototype, der fungerer som et spilbart produkt i forhold til at afprøve ideer, der potentielt skal overføres til den endelige version af spillet. Når man har afprøvet ideerne i en prototype, kan man reagere på feedback og lave endnu en iteration – og så videre indtil man har et ønskeligt resultat.

At arbejde med kørende software i samspil med en analysemetode som MDA gør også, at man kan implementere forskellige typer af test. I stedet for at identificere forskellige mechanics, dynamics og aesthetics og dele spillet op, så forskellige mennesker kan udvikle videre på forskellige dele, kan man i mange tilfælde i stedet arbejde ud fra kerne-mechanics og løbende – efterhånden som formodninger om dynamics og aesthetics afdækkes af tests – udbygge spillets kode. Det sikrer også, at koden kan testes på et teknisk niveau, og integration imellem forskellige mechanics kan sikres.

Som spiludvikler kan man altså med MDA først planlægge spillet, og så ved en iterativ, prototypebaseret udviklingsstrategi faktisk opleve spillet fra den anden side – nemlig ved som spiller af prototypen at undersøge aesthetcs, der kommer af de dynamics, man har med de givne mechanics.

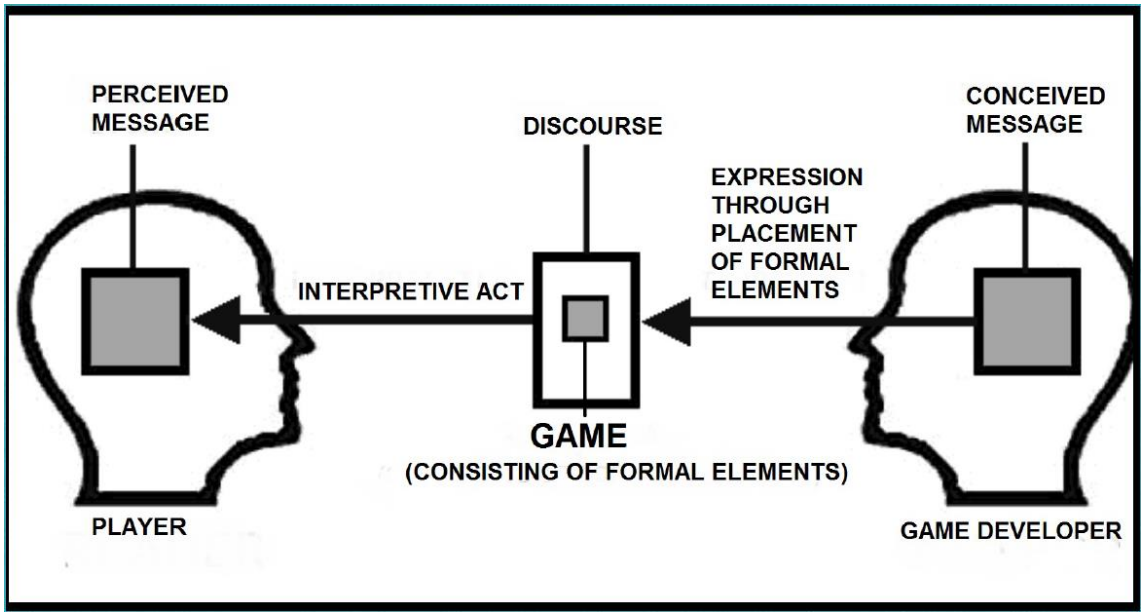
MDA-frameworket fungerer således bedst som praktisk værktøj, når det kombineres med iterationer, tests og prototyper.

Narrativitet

En måde at arbejde mere i dybden med de enkelte mechanics og dynamics er ved at anskue forholdet medialt. Det vil sige, at man opfatter computerspillet først og fremmest som et medie, der skal fortælle en række ting til spilleren.

Disse budskaber kan være alt fra hvordan en power-up skal bruges, til en forholden sig til spørgsmål om kærlighedens natur.

Hvis vi vil have dette fokus, kan vi med fordel bruge narrativitetsmodellen. Narrativitet dækker her ikke over en historie, men over hvad det er, der sker, når et budskab bevæger sig fra én hjerne til en anden igennem et medie. Her fra en spiludviklers hjerne til en spillers gennem et spil.



Vi ser her, at et budskab begynder som noget, en spiludvikler udtænker og som derefter udtrykkes igennem formelle elementer, der placeres i et spil. De formelle elementer sammen med en diskurs – den forståelsesramme, kulturen giver spillere, for eksempel at en sortklædt man, der snor sit overskæg mellem fingrene nok er ude på noget – informerer spillerens fortolkende handling. Den styrer, hvordan spilleren opfatter spillet. Det betyder, at hvad de formelle elementer betyder – og dermed hvilket budskab spilleren rent faktisk modtager – afhænger af mange flere faktorer end blot mechanics-dynamics sammenhængen kan afdække.

Modellerne i praksis

Lad os som et eksempel på, hvordan MDA og narrativitetsmodellen kan spille sammen se på det grundlæggende system i Doom. Man kan begynde med at sætte MDA-elementerne op i et skema for at få overblik:

Mechanics	• Man er en enlig soldat med begrænset liv, hvor hele verden er fjendtlig. • Banerne er bygget op som snoede gange. • Man kan samle skydevåben op og bruge dem som eneste interaktionsmulighed med andre væsner
Dynamics	• Spilleren bevæger sig igennem de snoede gange, samler våben op og skyder monstre • Spilleren kan dø, hvis monstrene skader ham eller hende nok
Aesthetics	• Spilleren føler sig som en super-marine, der kan klare alt • Spilleren føler sig isoleret og alene i en farlig verden • Spilleren føler sig forladt og ensom i dødsøjeblikket

Vi kan altså se de samme mechanics og de samme dynamics føre til tre vidt forskellige aesthetics. Denne kompleksitet kan narrativitetsmodellen bruges til at afdække. Vi kan for eksempel inddrage forskellige typer af spillere, der vil fortolke mechanics og dynamics (formelle elementer og til dels spillerens fortolkende handling) forskelligt. Føler man sig isoleret i virkeligheden, kunne der være større chance for at man føler sig isoleret og forladt i Doom. Vi kan også bruge narrativitetsmodellen til at pege på diskursen i Dooms samtid. Dengang hed det sig, at spil som Doom skulle gøre spillerne mere voldelige, fordi de fremmer vold som den eneste interaktionsform – en forklaring på de to første aesthetics meget modsat-rettede ordlyd.

En anden fordel ved modellen kan vises ved endnu et eksempel. I spillet Dys4ia bruges referencer til en lang række klassiske spil til i lighed med Doom at fremme en fornemmelse af isolation hos spilleren. For eksempel består en bane i, at spilleren skal styre en Tetris-brik igennem en åbning. Brikken passer bare ikke i åbningen! Banen skifter hurtigt, efter spilleren har fået et glimt af den isolerende oplevelse af ikke at kunne relatere sig til verden omkring sig.

Budskabet er på én måde det samme som i Doom, men fordi de formelle elementer er mere simple, bliver budskabet centreret omkring ensomheden – uden et tilsvarende andet budskab som Dooms super-marine. Samtidig er diskursen omkring Dys4ia – der spiller ind i en moderne transseksuel virkelighed – fundamental i at forstå, hvorfor Dys4ia ikke på samme måde som Doom kan medvirke til yderligere isolation hos spilleren.

Endelig kan vi vende tilbage til MDA-modellen og påpege, at der ikke er nogen mechanics, der omhandler brugen af vold til at hævde sig. Derimod er mechanics i Dys4ia centreret om individets plads i et omgivende, ensartet samfund snarere end kamp imellem forskellige enkeltindivider. Vi kan med andre ord mere nuanceret behandle aesthetics på tværs af genrer ved hjælp af en kombination af narrativitetsmodellen og MDA-modellen.

Ved hjælp af en kombination af MDA-frameworket og narrativitetsmodellen, kan vi analysere og planlægge vores spil tilstrækkeligt detaljeret til, at vi effektivt og målrettet kan lave prototyper.

Grundlæggende designspørgsmål for computerspil

Men hvornår ved man, om man har analyseret nok til at gå i gang med at prototype eller udvikle? Det er naturligvis to forskellige spørgsmål, men de grunder begge to i spørgsmålet: har du opnået nok forståelse af spillets formål til at kunne lave et produkt, der afspejler det?

For at besvare det spørgsmål, kan man gennemgå denne liste over grundlæggende designspørgsmål for computerspil. Hvis man har et rimeligt fornuftigt svar på dem alle sammen, er man klar til at starte udviklingen. Det vil sige, at man ikke skal have et udtømmende, entydigt og endegyldigt svar – men mere at man skal være i stand til at diskutere spørgsmålet fornuftigt i udviklergruppen og skrive tekst ned, der afspejler denne diskussion.

Er der nogle, man mangler svar på, og er analysen gået i stå, kan man kaste sig ud i en prototype, som undersøger netop de spørgsmål, man har afdækket her. Det giver mere retning i prototypearbejdet.

- Hvor stort er spillet?
    - Skal det tage 10 minutter eller 12 timer?
    - Hvor store er de enkelte sessions – tager det en time at komme ind i spillet, eller kan man samle det op og spille med det samme?
- Hvad laver spilleren i spillet?
    - Hvilke handlinger udfører spilleren?
    - Hvilke handlinger er de mest interessante for spilleren?
    - Hvilke handlinger bygger op til interessante handlinger?
- Hvad handler spillet om?
    - Hvad vil jeg sige med spillet?
    - Hvad siger de forskellige elementer i spillet og stemmer det overens med det, jeg vil sige?
- Spiller gameplayet ind på det, spillet handler om og hvordan?
    - Støtter spillerens handlinger op om det, du vil sige med spillet?
- Er spillet nyt?
    - Giver spillet en ny type oplevelse til spilleren eller er det en gentagelse af andre spil?
    - Hvilke elementer i spillet er nye / gentagelser?
    - Skal spillet være nyt?
    - Hvorfor eller hvorfor ikke?
- Hvad får man ud af at spille spillet, mere specifikt end at man er underholdt i så og så lang tid?

BUSINESS ANALYSE

Hvad sælger du og til hvem og hvordan?

Det kan lyde underligt, men det er faktisk sjældent helt klart for computerspiludviklere, hvad det er de sælger.

Størstedelen af computerspiludvikleres produkter er digitale. Det vil sige, at uanset om man sælger dvd'er eller via download, så er kerneproduktet i virkeligheden koden og de assets, koden udnytter – digitale billeder, digital lyd og så videre. Imidlertid er det et fåtal af kunderne, der rent faktisk bruger koden som kode eller assets som assets. Kunderne bruger det kørende spil, ikke bestanddelene. Man kan med andre ord sige, at i modsætning til klassiske produkter, som man får i hænderne og bruger, så sælger computerspiludviklere en interaktion eller en service. Man sælger ikke de ting, der er i spillet, men den mulige interaktion, spilleren får.

Problemet er imidlertid, at det er meget svært at se en interaktion og så lave den. Man laver typisk enkelte ting, som man opfatter som produkter. Dette stykke kode gør det her, denne 3d-model ser sådan ud og så videre, mens det er mere svært at få hold på, hvad interaktionen egentlig er. Med ord fra Content Analyse afsnittet kan man sige, at det er svært for udviklerne at se dynamics og aesthetics for bare mechanics.

Dette peger på den centrale udfordring for mange udviklere – de er fagfolk, der tænker som fagfolk, men har svært ved at sætte sig ind i deres kunders måde at bruge deres spil på. De har svært ved at svare på, hvem de sælger til, og derfor har de svært ved at svare på, hvad præcis det er, de sælger, for det de faktisk sælger er noget, der først opstår, når kunderne interagerer med kode og assets.

Kunderne køber et spil på løftet om, at det leverer en oplevelse til dem, som de er interesserede i. Derfor er det centralt at kunne svare på, hvem kunderne er, så man kan svare på, hvad kunderne finder interessant. Når man gennem sin content analyse har isoleret den oplevelse, man kan give med sit spil, kan man nemmere finde ud af, hvem man skal have fortalt, at denne oplevelse eksisterer.

Det vigtige er imidlertid, at man rent faktisk kan få sit spil frem til dem, der ville være interesserede i det. Hvis spil-oplevelsen svarer godt til et publikum, der primært bruger mobile touch-skærm enheder, men oplevelsen hænger uløseligt sammen med brugen af en mus, er det nærmest umuligt at placere spillet et sted, hvor det interesserede publikum kan bruge det. På samme måde som det ideelle sted at placere en limonade-stand er på en sommerdag ved siden af et sted, der sælger oversaltede popcorn, er der forskellige kunder, der holder til forskellige steder i spilinfrastrukturen. Det skal afgøres, hvor spillets kunder er, før man går videre.

Men det er ikke nok at finde ud af, hvor man skal sælge sit spil. Man må også spørge sig selv, hvordan man skal fortælle sine mulige kunder om oplevelsen. Det er ikke nok at lægge sit spil ud på en app store – man skal også have en måde at kommunikere til de mulige kunder, at det er der. Der uploades op til 2000 apps om dagen til for eksempel Apples App Store, så det er essentielt at finde ud af, hvordan man kommunikerer til sine mulige kunder.

Det vigtige i den forbindelse er at have identificeret to ting: hvor skal jeg sige noget for at ramme de rigtige og hvad skal jeg sige? Svarene på disse spørgsmål kredser i høj grad om, hvem kunderne er, og spiller sammen med content analysens svar på, hvilken oplevelse eller hvilket budskab, spilleren skal modtage.

Endelig er der en sidste overvejelse, der hænger tæt sammen med, hvem man sælger til. Nemlig hvor meget spillet skal koste. Der er forskellige standarder for pris afhængigt af produktets placering – mobilspil koster typisk mindre end PC-spil – og afhængigt af hvad man kan love og levere til kunderne – et 60 timers rollespil kan med rimelighed koste mere end et endless runner spil, der tager to minutter pr session – og endelig afhængigt af, hvor mange ressourcer man selv har brugt på spillet – hvad skal det koste, for at der ikke tabes penge på at lave spillet?

Til sammen kan man ordne disse overvejelser i en model, der dækker de grundlæggende spørgsmål inden for business analyse – den klassiske model de fire P'er.

- *Produkt:* Hvad består det, udvikleren leverer til kunden i?
- *Placering:* Hvor kan kunden få øje på, købe og eventuelt spille spillet?
- *Promotion:* Hvor skal man fortælle kunden om spillets eksistens og hvad skal man sige?
- *Pris:* Hvad skal spillet koste, når man tager produkt, placering og promotion med i overvejelserne?

Hvis man kan svare rimeligt fornuftigt og uddybende på disse fire grundlæggende spørgsmål, har man et godt udgangspunkt for at kunne få forretningsmæssig succes med sit spil eller sin spilvirksomhed.

Resten af dette afsnit vil præsentere nogle forskellige værktøjer, der kan hjælpe med til at uddybe og kvalitetssikre svarene på de fire grundlæggende spørgsmål inden for business analyse.

Hvordan markedsanalyserer man?

Svarene på de fire P'er afdækkes typisk i en overordnet markedsanalyse. Den skal organisere og sørge for indsamling af viden omkring, hvordan markedet er for den type produkt, man leverer. Herunder skal der svares på spørgsmål som om der er lignende produkter, om der er stærke konkurrenter, hvordan andre produkter i samme kategori markedsføres eller hvad den gængse pris er. Men hvordan laver man sådan én? Den sikreste metode er ved netop at tage udgangspunkt i

modellen med de fire P'er og formulere spørgsmål og kilder til svar. Det kan sættes op i skema som dette:

	Spørgsmål	**Svarkilder**
Produkt		
Placering		
Promotion		
Pris		

Man kan enten lave generelle spørgsmål og svar, eller specifikke spørgsmål og svar, der passer til lige præcis det produkt, man undersøger. Det er i sagens natur svært at komme med dækkende eksempler på specifikke spørgsmål og svar, men her er nogle generelle, der passer til de fleste spil:

	Spørgsmål	**Svarkilder**
Produkt	1. Hvilken oplevelse får spilleren? 2. Hvilke andre spil giver samme oplevelser? 3. Hvilke andre spil ligner produktet grafisk? 4. Hvilke andre spil ligner produktet gameplaymæssigt? 5. Hvilke spil er produceret af lignende firmaer?	1. Content Analyse, prototyper, playtests 2. Content Analyse af lignende spil, der også behandles under andre P'er 3. Content Analyse af lignende spil, der også behandles under andre P'er 4. Content Analyse af lignende spil, der også behandles under andre P'er 5. Udvælgelse af firmaer, der matcher egen størrelse, geografiske placering og/eller erfaringsniveau
Placering	1. Hvilke platforme egner spillet sig til? 2. Hvilket publikum er der på de udvalgte platforme? 3. Hvilken infrastruktur er der til at nå publikummet? 4. Hvilke lokaliseringsudfordringer er der?	1. Content Analysen af spillet 2. Kvantitativ eller kvalitativ data fra troværdige kilder såsom salgsopgørelser, analyser af anmeldelser eller fokusgrupper 3. Spiludviklere der kender det lokale marked man vil ind på, app store-udbydernes kvantitative data 4. Spiludviklere der kender det lokale marked man vil ind på, oversættere

Promotion	1. Hvilke kanaler er der til publikummet? 2. Hvad skal publikummet loves? 3. Hvordan virker forskellige typer af markedsføring på publikummet?	1. Kvantitativ eller kvalitativ data om publikummets medievaner, kortlægning af udviklernes netværk 2. Content Analysen, feedback fra playtests 3. Spiludviklere der kender det lokale marked man vil ind på, kvantitativ eller kvalitativ data om publikummets medievaner
Pris	1. Hvad er den typiske pris i infrastrukturen? 2. Hvad er den typiske pris for spiltypen? 3. Hvad er prisniveauet hvis der skal opretholdes en profitmargen?	1. Kortlægning af forskellige prisniveauer i app stores/fysiske butikker 2. Sammenstilling af lignende spil, der også behandles under andre P'er 3. Budget, der viser, hvad udgifterne for udviklingen af spillet er og hvor break-even punktet er

Med en oversigt som denne kan man undersøge, om man har dækket alle de afdelinger af business analysen, der er nødvendige for at man har et oplyst grundlag, og man kan planlægge, hvordan man skal indhente oplysningerne. I en færdig oversigt vil man formodentlig tilføje eksplicitte hjemmesider, datakilder og kontakter, så det er nemt at komme til svarkilderne.

Skal man bruge skemaet til projektstyring kan man tilføje en ekstra kolonne, der kortlægger, hvem i firmaet/teamet der har ansvaret for at svare på hvilke spørgsmål, så skabelonen ser sådan her ud:

	Spørgsmål	**Svarkilder**	**Ansvarshavende**
Produkt			
Placering			
Promotion			
Pris			

En del af de svarkilder, der er nævnt i eksemplet oven for refererer til kvantitativ og kvalitativ data, og det er værd lige at standse op et øjeblik og reflektere lidt over disse to datatyper og deres forhold til hinanden.

Kvantitative og kvalitative undersøgelser

I forhold til at undersøge og beskrive verden, er det værd at huske på denne lille novelle (*Del rigor en la ciencia)* af Jorge Luis Borges, her i engelsk oversættelse af Andrew Hurley:

"In that empire, the art of cartography attained such perfection that the map of a single province occupied the entirety of a city, and the map of the empire, the entirety of a province. In time, those unconscionable maps no longer satisfied, and the cartographers guilds struck a map of the empire whose size was that of the empire, and which coincided point for point with it. The following generations, who were not so fond of the study of cartography as their forebears had been, saw that that vast map was useless, and not without some pitilessness was it, that they delivered it up to the inclemencies of sun and winters. In the deserts of the west, there are tattered ruins of that map, inhabited by animals and beggars, in all the land there is no other relic of the disciplines of geography."

Hvis man forsøger at beskrive virkeligheden, bliver beskrivelsen – som man ofte kalder en hypotese, altså en påstand man har om, hvordan verden ser ud – aldrig fuldstændig 1:1 perfekt. Det væsentlige er at balancere imellem at reducere en beskrivelse til det praktisk nødvendige og stadig få et retvisende billede med. Man skal med andre ord kunne navigere efter kortet, men om hver eneste vej eller lille bæk er med på det er mindre væsentligt.

Der findes to overordnede videnskabelige metoder til at efterprøve hypoteser, på en måde, der som regel kan overkomme denne balanceudfordring: kvantitative og kvalitative.

Den kvantitative metode består i at forsøge at reducere verden til tal, der kan sammenlignes med hinanden umiddelbart. Hvis man for eksempel spørger sig selv, hvilket af to spil der er mest succesfuldt, så kan man sammenligne salgstal for dem begge to. Det er der ikke noget galt i, men man skal være opmærksom på, hvad man skærer væk. Her reducerer man det, at et spil er succesfuldt til en bestemt del af spillet, nemlig om det er finansielt succesfuldt. Man kan også sammenligne Metacritic scores for at få en idé om, hvilket spil der blev modtaget bedst af anmelderne, eller brugerratings på Metacritic for at forsøge at få adgang til andre spilleres indtryk. Man kan også med mange forskellige, teknologisk baserede analytiske metoder indsamle en del kvantitativ data om hvordan spillere spiller ens spil, for eksempel hvor mange spillere der når forbi et bestemt puzzle.

Fordelene ved den kvantitative metode er, at hvis man udvælger sine taltyper med omhu, får man en nem måde at sammenligne forskellige spil eller løsninger i designet af spil på. Det er for eksempel meget retvisende at sammenligne mængden af brugere af Google Play Store i Hong Kong med mængden af brugere i Apples App Store samme sted for at vælge det bredeste marked.

Man skal imidlertid passe gevaldigt på at lade den kvantitative metode stå alene. Forsøger man for eksempel at lave en ændring i et puzzle ud fra den kvantitative metode – altså hvor mange spillere, der giver op – risikerer man at ødelægge den fornemmelse af sejr, der kan opstå hos de spillere, der kommer igennem puzzlet. Fornemmelser, oplevelser, interaktivitet – alle kernedelene af spil som produkt – er svære at repræsentere præcist ved hjælp af kvantitativ data.

Her kommer den kvalitative data ind. Hvor den kvantitative data oftest kan indsamles gennem standardiserede spørgeskemaer, salgsrapporter eller automatiske overvågningssystemer bygget ind i spilkode, kommer kvalitativ data fra at man snakker med mennesker og dokumenterer, hvad de sagde. I stedet for at logge, hvad spillere gør i et spil, spørger man dem, hvordan det oplevedes at gøre det.

Kvalitativ data forsøger således at kortlægge kompleksiteten i menneskers interaktion med spillene ved at have samtaler, der ikke skal reduceres til tal. Skal man sammenligne med anmeldelser, er kvalitativ data på mange måder hele den fulde tekstanmeldelse over for kvantitativ data som den opsummerende score.

Det betyder selvfølgelig ikke, at kvalitativ data bare genereres ved at man sådan snakker lidt om tingene med nogle spillere. Man forsøger at systematisere ved at stille nogenlunde enslydende, åbne spørgsmål til alle interviewpersoner, og man forsøger at få et bredt indblik i forskellige spilleres reaktioner ved at interviewe folk med forskellig baggrund.

Igen kan den kvalitative metode imidlertid ikke stå alene. Det primære problem er, at netop når dataen kan være så forskellig, som forskellige menneskers mere eller mindre ordrige svar på nogle spørgsmål snarere end forskellige tal, kan det være svært at sammenligne dataen. Det kan gøre det problematisk at komme med konklusioner. Samtidig er det praktisk umuligt at gennemføre en tilstrækkeligt repræsentativ undersøgelse, når man ikke kan automatisere dele af processen.

Hvad gør man så? Man forsøger at kombinere de to forskellige datatyper og bruge deres styrker i forhold til hinanden, så man kan krydstjekke de konklusioner, man kommer med på baggrund af den ene datatype med input fra den anden.

Et eksempel på, hvordan man kunne gøre dette er:

1. Udstik en hypotese
2. Efterprøv hypotesen med kvantitativ data, for at sikre at den ikke er skudt helt ved siden af
3. Juster hypotesen
4. Efterprøv den nye hypotese med kvalitativ data
5. Juster hypotesen
6. Planlæg og ager ud fra hypotesen

Eller hvis man skulle lave et eksempel:

1. Hypotese: Typiske spillere vil betale 72 kroner for et endless runner spil på iOS

2. Saml priser på endless runner-spil til iOS sammen samt demografisk information om typiske iOS-spillere
3. Juster hypotesen: Typiske spillere vil betale mellem 0 og 2.99 dollars for et endless runner spil på iOS
4. Efterprøv hypotesen ved at tale med fem personer, der er udvalgt for at dække den typiske demografi, der spiller på iOS
5. Juster hypotesen: Typiske iOS-spillere mellem 15-25 vil betale 0 dollars/kroner for et endless runner spil
6. Find en forretningsmodel, hvor folk ikke skal betale for spillet

Læg mærke til, at vi på intet tidspunkt i løbet af de seks trin beskrev virkeligheden præcis som den var, men langsomt og systematisk kom tættere og tættere på en konkret virkelighed. Vi startede med vores forestilling, justerede den med kvantitativ data, justerede denne med kvalitativ data og endte med en plan, som vi er rimeligt sikre på vil virke i forhold til et stort udsnit af mennesker, som ligner dem, vi har haft tæt kontakt med i den kvalitative undersøgelse. Det er styrken ved kombinationen af de forskellige datatyper.

Release-strategi

Men når man har besluttet sig for, hvad man skal sige og til hvem, kommer man frem til den problematiske beslutning om, hvordan man skal promovere sit spil. Igen er dette et sted, hvor mange udviklere har problemer, idet der er en fornemmelse af, at hvis man laver et godt spil, så er det nok at gøre opmærksom på det en enkelt gang, så skal det nok klare sig selv.

Dette er ikke korrekt.

Mere end noget andet sted i spiludviklingens business-afdeling, er der her brug for udholdenhed. Hvis man ved, hvad man skal sige og til hvem, så skal man gentage det budskab igen og igen og igen.

Det betyder ikke, at man skal spamme folk med de samme statusopdateringer hele tiden, men at man skal finde forskellige måder konstant at sige det samme på alle kanaler.

Et godt eksempel på denne markedsføringsstrategi er kampagnen for Goat Simulator.

Spillet er et bevidst dårligt lavet spil, der på samme måde som med en B-film skal nydes sammen med vennerne over nogle bajere. Man styrer en ged, der render rundt i en lille, amerikansk by og kan gennem spillets fejl-fyldte interaktion få ting til at eksplodere og lave mærkelige tricks med geden. Det grundlæggende budskab er: geder er sjove, mærkelige og absurde; det er skægt at få geder til at lave underlige ting; at spillet er en parodi på mere seriøse, tredje-persons action og simulator-spil.

Disse tre budskaber bliver kommunikeret i trailers, sociale medie opdateringer, nye opdateringer af spillet – i skrivende stund med en parodisk zombie-update, GoatZ, der gør grin med survival simulationsspil som DayZ – og endda i spilstudiets community management. Her er tommelfingerreglen, at man må tage så meget pis på kunderne, man vil, så længe det er sjovt. Dette gælder selv under support.

Det gælder simpelthen om at være skarp i forhold til hvad oplevelsen er for spilleren og så være endnu skarpere til at formulere denne oplevelse igen og igen på tværs af forskellige platforme og udtryksformer. Det vigtige er ikke at hver enkelt formulering er det mest fantastiske, der nogensinde er blevet formuleret, men derimod at den rammer det rigtige budskab nogenlunde og at der bliver ved med at være et rimeligt flow af budskaber. Det betyder ikke, at der nødvendigvis skal være mange daglige updates – selv om det absolut ikke er en dårlig idé - men at der skal være et konstant pace. Én video om dagen, et multi-tweet twitter rant om dagen, en Facebook opdatering om dagen, én blog om ugen – så længe man kan regne med, at den kommer konsistent og der er et acceptabelt bundniveau, så er aktivitet klart at foretrække frem for kvalitet.

Det er på makroniveau, at man skal udvælge sig nogle faste intervaller for sin kommunikation. Men på mikroniveau, hvornår er det så en god idé at kommunikere det ud? Langt det meste kommunikation et lille udviklerhold kan overskue at arbejde med vil være på sociale og digitale medier. De bedste tidspunkter at dele sit budskab på sociale medier er mellem 07:00-08:30 og igen mellem 15:00-17:00. Der er folk enten under opstart med deres arbejde eller ved at lukke det ned, og der er der mest chance for, at de bruger tid på at læse og sprede budskaber. Går man efter et andet marked end Danmark gælder det samme i princippet, men det kan være en fordel at sætte sig ind i lokale arbejdsforhold.

Det er svært at lave en fuldstændig plan over en releasestrategi, men det nedenstående skema kan måske hjælpe til at sætte nogle overvejelser i system. Det er sporadisk udfyldt med et eksempel, der involverer fodboldspillet Fifa:

Platform	**Budskab**	**Interval**	**Brainstorm af forskellige formuleringer**
YouTube	Man kan spille med de rigtige ligaer	Hver uge, søndag	Let's Plays med de forskellige ligaer, der matcher de igangværende i virkeligheden
Facebook	Fifa er perfekt til en hyggeaften med vennerne	Hver uge, onsdag, i 4 uger før og fire uger efter release	Konkurrence på Facebook, hvor man kan vinde en ekstra controller og det nye Fifa-spil
Twitter	Man kan spille med de rigtige ligaer	Konstant	Interaktion med brugerne om fodbold-trivia
Snapchat	Man kan spille med de rigtige ligaer	Hver dag	Dagligt snap med nye screenshots af højt profilerede hold

Et sådant skema kan give overblik over, hvilke budskaber, der er i spil og om de er dækket på forskellige platforme. Man skal sikre sig, at budskaberne ikke overlapper hinanden, men supplerer hinanden sådan så der altid er en ny udformning af de budskaber, man har, i folks feed.

Endelig kan det være fornuftigt at lave noget så gammeldags som en plan med datoer. Hvad sker der på hvilke tidspunkter i løbet af releasen? Hvornår er hvilke budskaber i højsædet? Et strategisk skift i hovedbudskab kan være med til at forlænge et produkts nyhedsværdi betydeligt.

Hvornår skal man så gå i gang med sin releasestrategi? I virkeligheden så hurtigt som muligt. Tweet og screencast gerne under udviklingen, så længe det gøres ofte og regelmæssigt. Det opbygger langsomt et community, der bliver vant til at lytte til budskaber fra spilfirmaet, så de er klar, når det vigtigste budskab kommer: køb spillet!

Forretningsmodeller

Vi har indtil videre koncentreret os om enkelte produkter og hvordan man gør sit forarbejde i forhold til dem. Hvis man vil have en levedygtig spilvirksomhed er det imidlertid kun den ene del af spørgsmålet. Et succesrigt produkt skal helst passe ind i en overordnet plan for, hvordan man kan blive ved med at levere succesrige produkter igen og igen. Herunder skal man løbende lære mere og mere, der kan give bedre svar på de fire P'er i fremtidige produkter.

Konkrete forretningsplaner udvikler sig meget hurtigt i digitale brancher som spilindustrien, og derfor er det svært at se på succeshistorier og lære noget af dem, sådan som man ellers typisk gør, når man udvikler forretningsplaner. Ikke desto mindre er der en række forretningsmodeller, der i løbet af en årrække har konsolideret sig som centrale måder at indrette sin forretning på, specielt i en skandinavisk sammenhæng. De er kortlagt i en udgivelse fra EU-projektet Scandinavian Game Developers med titlen The Business of Making Games og omfatter:

- *Selling a Product:* at man sælger spillet som ethvert andet produkt
- *Freemium:* at man giver dele af spillet gratis for at kunne sælge resten
- *Work for Hire:* at man producerer spil ikke til markedet men til bestemte klienter
- *Revenue Through Advertising:* at man sælger reklameplads i sine spil
- *Funding:* at man finder stater eller private, der vil investere i ens firma

Der er også inkluderet et afsnit om de strategiske overvejelser i forbindelse med forskellige typer af oplevelser, man kan levere til sine kunder, primært med fokus på underholdning over for andre typer af oplevelser så som læringsspil eller politiske spil.

Udgivelsen med uddybende beskrivelser kan hentes gratis her: kortlink.dk/guug

De fire P'er

En stor del af business delen af at være en succesrig spiludvikler er ikke særlig kompliceret, men kræver til gengæld en del fodarbejde. De ovenstående afsnit skulle gerne have givet en fornemmelse af nogle af udfordringerne, der møder spiludviklere, der gerne vil skabe forretninger og gøre spiludvikling til deres levebrød. Som nævnt kan mængden af arbejde på dette område nemt vokse en udvikler, der egentlig bare ville lave spil over ørerne. I sådanne situationer skal man huske på, at så længe basis-analysen er stærk, så kan man navigere rimeligt effektivt. Svar på de fire P'er:

- *Produkt:* Hvad består det, udvikleren leverer til kunden i?
- *Placering:* Hvor kan kunden få øje på, købe og eventuelt spille spillet?
- *Promotion:* Hvor skal man fortælle kunden om spillets eksistens og hvad skal man sige?
- *Pris:* Hvad skal spillet koste, når man tager produkt, placering og promotion med i overvejelserne?

Hvis aktiviteterne inden for business-området hele tiden har for øje, at de skal referere til disse fire principper, kan selv meget tunge processer styres relativt sikkert i havn.

Gangen i OOA

Objektorienteret Analyse er en holistisk disciplin. Derfor er det vigtigt at have en klar fornemmelse af gangen i en analyse.

Her er en kort oversigt.

Systemafgrænsning
Kommer af: Content- og Marketing-analyser
Leder til: Use Cases

Use Cases:
Kommer af: Systemafgrænsning, content- og marketing-analyse
Leder til: Objekter, klasser, hændelser, hændelsestabel, funktioner, system definition

Objekter og klasser:
Kommer af: Use cases
Leder til: Hændelsestabel, klassediagram

Hændelser:
Kommer af: Use cases
Leder til: Hændelsestabel

Hændelsestabel:
Kommer af: Klasser og hændelser
Leder til: Klassediagram, funktioner

Klassediagram:
Kommer af: Hændelsestabel
Leder til: Klassediagram i design-fasen, systemdefinition

Funktioner:
Kommer af: Hændelsestabel, use cases
Leder til: Systemdefinition, metoder i design-fasen

Systemdefinition:
Kommer af: Use cases, klassediagram, funktioner
Leder til: design-fasen

OBJEKTORIENTERET ANALYSE

Hvad er objekter og hvorfor hjælper de udviklere?

Den måde langt de fleste stykker software er bygget op på er gennem objektorienteret tankegang. Det vil sige at frem for at tænke for eksempel hver eneste knap i et tekstbehandlingsprogram som sin egen unikke ting, der skal konstrueres for sig, samler man nogle fælles funktioner i én konstruktion og genbruger den. For eksempel skal alle knapper kunne trykkes på, de skal have et ikon, og de skal have den samme grafiske effekt, når man kører markøren hen over dem. Man sparer således meget udviklerarbejde ved at analysere sig frem til generelle, samlede funktioner, f.eks. en knap – klasser – som man udmønter i specifikke funktionaliteter, f.eks. gem-kanppen – objekter.

Hvordan laver man så en objektorienteret analyse med fokus på spil?

Et rimeligt spørgsmål, som folk fra spiluddannelser har stillet sig selv i mange år. Der er mange fornuftige ressourcer til at finde svar på det, hvis man er normal softwareudvikler. Men hvorfor og hvordan laver man sådan en analyse, hvis det man vil udvikle er spil?

Hvorfor er et spørgsmål om klarhed. Man kan bruge OOA-værktøjerne til at skabe en klarhed over de forskellige dele af et spil og til at beskrive gameplay på en måde, der er mere systematisk end man kan med ord. Denne klarhed kan derefter veksles til overblik og god kommunikation i en udviklingssituation.

Nedenfor er en kort gennemgang af, hvordan man bruger de forskellige OOA-værktøjer, og hvordan de spiller sammen. Netop sammenspillet er essentielt at forstå, fordi det gør en i stand til at følge en tanke fra den opstår til den er realiseret

og bære ideer igennem en hel udviklingsproces. Værktøjerne er skrevet op i den typiske rækkefølge, de fremtræder i skriftlig dokumentation. Den afspejler også én meget brugbar og anbefalet rækkefølge i udførelsen af analysen.

Systemafgrænsning

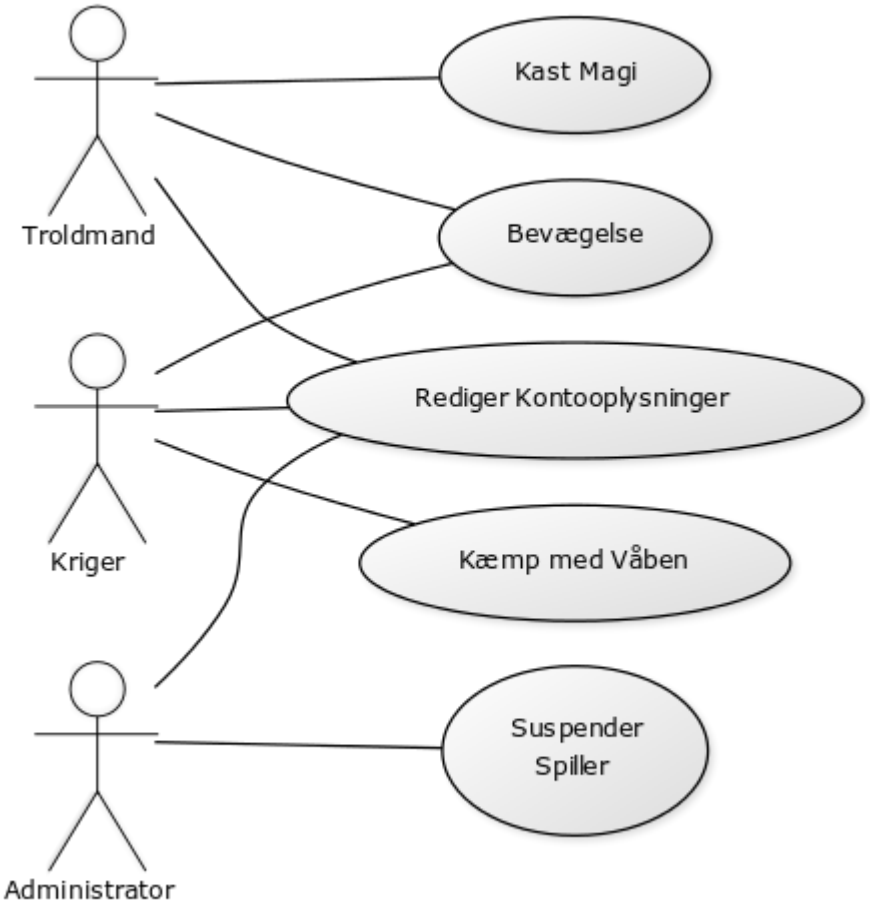
En meget, meget kort beskrivelse af systemets kerneydelse og indretning med udgangspunkt i, hvilken rolle det skal spille i brugerens hverdag. Er en udløber af de separate analysefelter content og marketing.

Systemafgrænsningen skal komme ind på spillets genre, en enkelt core mechanic og et muligt publikum og platform. F.eks.: ”Palnatoke Wars er et turbaseret strategispil om vikingernes invasion af England. Kernen af spillet er at håndtere logistikken i at flytte store mængder soldater over havet og rundt omkring på fremmed jord. Det appellerer til historisk interesserede strategi-nørder og har primær platform på PC med sekundær mulighed for IOS.”

Use Cases

En use case er en detaljeret beskrivelser af en afgrænset interaktion med systemet/spillet. Ikke én samlet use case over hele systemet/spillet, men heller ikke én use case per knap, man trykker på. I for eksempel Fifa vil man typisk ikke have en use case, der beskriver stikafleveringer, men en use case, der beskriver alle de forskellige mulige afleveringer.

Der kan gives et overblik i et Use Case Diagram, hvor det vises, at forskellige typer af spillere – actors – har adgang til forskellige use cases. For eksempel har en spiller i et typisk rollespil, der spiller en troldmand, adgang til andre use cases end en spiller, der spiller en kriger. Samtidig kan andre individer uden for spillet have adgang til bestemte use cases, for eksempel en administrator i et online rollespil, der kan suspendere brugere og redigere deres kontooplysninger, men ikke deltage i spillet. Her er et simpelt eksempel:



Der skal gives detaljer for hver enkelt use case i en use case tekst. Denne er ofte formuleret i en skabelon. Der eksisterer en del af dem, men på Dania Games har vi lavet vores egen – baseret på Alistair Cockburns[1] – der er rettet imod spiludvikling.

Der er i tilgift også et par valgfri punkter, der måske/måske ikke kan være behov for.

[1] http://alistair.cockburn.us/get/2465

Use Case Section	Comment
Use Case Name	Vælg et beskrivende, selvindlysende navn; start gerne med et udsagnsord
Description	En kort beskrivelse af hvad use casens mål er i almindeligt sprog
Actors	Hvilke actors – det vil sige, hvilke handlende individer uden for spillet – er involveret i use casen?
Preconditions	Hvilke ting skal være opfyldt før use casen kan startes
Main Success Scenario	Hvordan er processen igennem use casen, hvis intet går galt. Kan med fordel inddeles i nummererede skridt, der kortlægger input fra spilleren og output fra spillet
Alternative scenarios	Alternative veje igennem use casen. Kan både indeholde succesveje og veje, hvor fejl opstår
Miscellaneous	Diverse andre bemærkninger, der kan være nyttige for senere udviklere

Use Case Section	Valgfrie punkter
Frequency of Occurrence	Hvem aktiverer use casen? Bliver den brugt tit eller sjældent?
Technology and data variations list	F.eks. I/O interaktion, specielle data formater m.v.

Use cases opstår fra de brugermæssige behov, der er afdækket i de foregående analyser, der i use cases skal gøres konkrete og beskrives så minutiøst som muligt. På den måde danner brugerens behov – via use cases – udgangspunkt for systemets indretning, da alt følgende OOA udspringer fra use cases.

Typisk er use cases også udgangspunktet for tests, der dog klassisk set ligger i implementeringsfasen, men også kan forekomme efter man har udarbejdet en prototype.

Objekter + Klasser

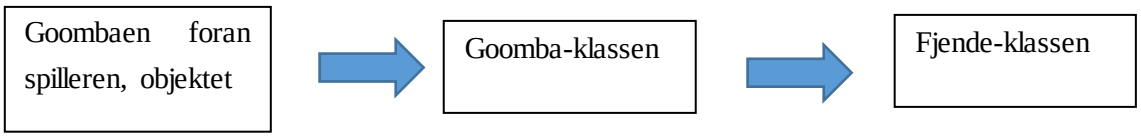
Når man har gjort sig det klart, hvilke interaktioner, spilleren har med spillet, kan man trække en lang række forskellige objekter ud af de beskrivelser, man har inkluderet i sine Descriptions, Actors og Scenarios.

Objekter dækker i OOA alle de ting, en spiller kan se på skærmen og som spilleren i øvrigt interagerer med.

Typisk vil et spil være bygget op af en lang række objekter, der har meget til fælles – for eksempel er Koopa Troopers, Goombaer, Thwomps og så videre fra Super Mario-spillene alle sammen fjender, men hver type fjende fremstår forskellig. Vi organiserer derfor typisk disse objekter i klasser, der er en generaliseret form for objekt.

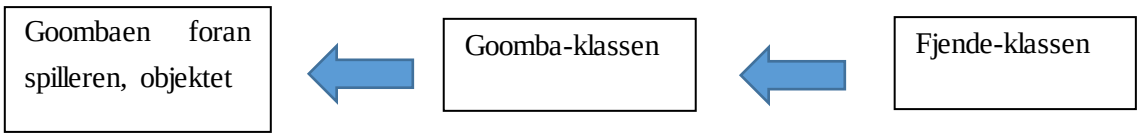
Det gør vi, fordi den senere kode kører mere optimalt, hvis man kan indlæse fælles kode fra ét sted i systemet – klassen 'fjende' – og så nuancere det med information fra en underklasse – for eksempel Goomba – og endelig vise et eller flere eksemplarer af underklassen frem på skærmen – den Goomba, der står over for spilleren netop nu.

Det vil sige, at vi i analysen forestiller os objektet, generaliserer det op i en klasse, for til sidst at reducere antallet af klasser mest muligt og samle i generelle klasser. Et billede på denne analytiske bevægelse kunne se sådan her ud:



Vi ser Goombaen og generaliserer ud fra den, at der må være en typisk Goomba med fælles Goomba-træk. Selv om én Goomba for eksempel starter oppe på en platform, mens en anden starter under den, går de begge to fra højre mod venstre og falder ud over kanten af platforme. Den typiske Goomba deler en del træk også med andre objekter, der ikke er Goombaer, men alle er fjender for Mario, for eksempel at de kan skade Mario. Derfor generaliserer vi en fjende-klasse, som Goomba-klassen er en specialisering af.

Det var den analytiske bevægelse. Men sådan her ser det faktisk ud, når systemet kører:



I systemets kode skal der ligge en fjende-klasse, der giver en række informationer og fællestræk til Goomba-klassen, der instantierer hvert enkelt Goomba-objekt, der af systemet kan køres uden at læse forskelligt kode ind for hver enkelt Goomba. I stedet kan systemet blot kopiere informationen, der er fælles og så variere for hvert enkelt Goomba-objekt.

Fjende-klassen kaldes en superklasse, mens Goomba-klassen kaldes en sub-klasse. Det forhold, at sub-klassen automatisk inkluderer informationer fra en fjende-klasse kaldes nedarving. Man siger, at Goomba-klassen arver en række informationer og funktionaliteter fra fjende-klassen.

NB: De to ovenstående figurer er ikke klassediagrammer, men skal blot illustrere tankegangen.

Når man skal beskrive objekter og klasser, gør man det i første omgang ved hjælp af det, man kalder en rig klasseliste. Man laver en ren tekstliste, der fungerer som brainstorm over alle de mulige klasser, der kunne anvendes for at realisere de use cases, man har stillet op tidligere.

Man behøver ikke låse sig fast på en endelig klasseliste her, for med værktøjet Hændelsestabel kan man sortere i klassekandidaterne og finde frem til dem, der rent faktisk er brug for, for at systemet/spillet kan køre.

Endelig danner den rige klasseliste udgangspunkt for indholdet i klassediagrammet.

Hændelser

Mens objekter og klasser beskriver de ting, der er i et spil/system, beskriver hændelser de ting, der sker. Hændelser skal tænkes som fuldstændig klare punkter i tid, hvor der er en overgang fra én tilstand til en anden.

Det vil sige, at hvor man kan tale en multiplayer match i et First Person Shooter-spil som et hændelsesforløb, vil man ikke kalde det for en hændelse. En hændelse i den kontekst vil typisk være sådan noget som ”skyd”, ”saml våben op”, ”liv mistes”, ”fjende rammes” og så videre. Et komplekst forløb i et spil vil skulle beskrives som en lang række af hændelser, der ud fra et systemanalytisk synspunkt ikke hænger sammen, men er forskellige hændelser, der sker i systemet.

Det vil sige, at vi forestiller os et hændelsesforløb, men isolerer de forskellige enkelte hændelser, der sker i det, for at kunne beskrive klart, hvilke forandringer, systemet skal understøtte i sig selv. Det er med andre ord systemets dynamik, vi her beskriver.

Når vi spiller en soldat, der løber hen og samler et våben op, som soldaten skyder en anden soldat med, hvorefter den anden soldat dør og den første soldat får point, vil vi således ikke forsøge at beskrive det flow, vi observerer, men i stedet de enkelte hændelser, der sker. ”Løb”, ”saml våben op”, ”skyd”, ”fjende rammes”, ”fjende dør” – sådanne hændelser skal kunne ske i systemet.

På samme måde som objekter og klasser bevæger vi os altså fra et specifikt til et generelt niveau, når vi taler hændelser. Ikke hvad der sker i den enkelte interaktion, men hvad der kan ske generelt i systemet. Vi beskriver paletten af mulige hændelser, ikke hvordan de skal sættes sammen.

Når man skal beskrive hændelserne gør man det i første omgang ved hjælp af en rig hændelsesliste. Det vil sige, at ligesom med klasser laver man en tekstbaseret liste over alle de mulige hændelser, man kan se systemet kan have brug for.

Hændelserne genreres ud fra use cases, der jo beskriver mere sammenhængende interaktioner.

Hændelserne skal forholde sig til den rige klasseliste, forstået på den måde, at vi ved at danne os et overblik over, hvad der kan ske i systemet også kan afsløre, om vi har glemt nogle ting, hændelserne kan ske for. Det bliver vigtigt i hændelsestabellerne, der sammenstiller klasselisten og hændelseslisten.

Hændelsestabel x 2

En hændelsestabel er sammenstillingen af den rige klasseliste og den rige hændelsesliste.

Tabellen skal vise, hvilke hændelser, der kan ske i systemet og hvilke klasser, der er involverede i hvilke hændelser. Altså: hvad kan der ske og hvem/hvad sker det for?

Sådan her ser det ud:

	Klasse 1	Klasse 2	Klasse 3
Hændelse A	X	X	
Hændelse B		X	
Hændelse C	X		X

Vi kan her se, at når A sker i systemet, involverer det objekter, der er instantierede fra klasserne 1 og 2. De klasser kan selv være med til at påbegynde hændelsen, eller de kan simpelthen blot blive påvirket af den

Med FPS-eksemplet fra ovenfor vil det se sådan her ud:

	Soldat 1	Soldat 2	Våben
Løb	X		
Saml våben op	X		X
Skyd	X		X
Fjende rammes	X	X	X
Fjende dør		X	

Men denne hændelsestabel dækker kun den enkelte situation, vi har beskrevet. Soldat 2 kan formodentlig også bevæge sig, samle våben op og skyde, så i spillet som sådan ville man kunne stille det op således:

	Soldat 1	Soldat 2	Våben
Løb	X	X	
Saml våben op	X	X	X
Skyd	X	X	X
Fjende rammes	X	X	X
Fjende dør	X	X	

Dette er den første type af hændelsestabel, vi konstruerer. Den viser umiddelbart alle de klasser, der er på vores rige klasseliste ud fra den beskrevne interaktion. Men der er problemer med den.

For det første er Soldat 1 og 2 inkluderet i alle hændelser. Dette er typisk et tegn på, at man har konstrueret sit system på en måde, så det er meget afhængigt af en enkelt klasse. Inden for produktivitetsapps er dette sjældent en god idé, da et system oftere skal stille en række forskellige værktøjer til rådighed for en bruger, der så er den samlende kraft, men som ikke er repræsenteret i systemet. Tænk på et tekstbehandlingsprogram, hvor der ikke som sådan er en repræsentant for brugeren i selve systemet – kun værktøjer til at indføre data.

I et spil er der imidlertid typisk en repræsentation af spilleren på skærmen – en avatar – og denne repræsentation vil ofte være involveret i alle eller nærmest alle hændelser. Gør gerne i kommentarer til hændelsestabellen opmærksom på dette specielle forhold for spil.

Det andet problem er, at Soldat 1 og Soldat 2 er præcis ens. Det tyder på, at der ikke er brug for en klassedifferentiering imellem dem. Når to klasser er involveret i præcis de samme hændelser, er det et godt fingerpeg om, at man kan lave en generaliseret superklasse i stedet.

Man vil typisk reagere på sådanne problemer med at lave en ny hændelsestabel, der viser en mere hensigtsmæssig konstruktion. Den kunne se sådan her ud:

	Soldat	Våben
Løb	X	
Saml våben op	X	X
Skyd	X	X
Fjende rammes	X	X
Fjende dør	X	

Denne hændelsestabel afdækker en række hændelser, der kan ske i systemet og hvilke klasser, der er involveret i disse forskellige hændelser. Kampen i eksemplet oven for kan således forstås som et sammenstød mellem to instantierede objekter af superklassen Soldat – de enkelte objekter har kun kosmetiske forskelle – hvor hændelserne Løb og Fjende dør kun involverer de to objekter, mens hændelserne Saml våben op, Skyd, Slå med kniv og Fjende rammes i tilgift involverer et objekt af superklassen Våben og dens sub-klasser.

Det betyder, at vi har overblik over, hvilke bestanddele, der skal til for at levere en bestemt funktionalitet, der kan støtte op om en lang række interaktioner, blandt andet den i vores eksempel.

I forhold til et fuldt, avanceret FPS-spil, vil man næppe kunne nøjes med én klasse til alle spillere. Der er forskellige typer af soldater – nærkamp, mellemkamp, snipere – der formodentlig netop karakteriseres ved at de har adgang til forskellige hændelser i spillet.

Lad os forestille os, at der er en maskingeværssoldat og en nærkampssoldat. Maskingeværssoldaten kan skyde, mens nærkampssoldaten kun kan bruge en kniv. De kan hver især samle våben op, uanset om det passer til deres type eller ej. Så ville hændelsestabellen se sådan her ud:

	Superklasse: Soldat	Subklasse: Maskingevær-soldat	Subklasse: Nærkamps-soldat	Superklasse: Våben	Sub-klasse: Maskingevær	Sub-klasse: Kniv
Løb	X					
Saml våben op	X			X		
Skyd		X			X	
Slå med kniv			X			X
Fjende rammes på afstand	X				X	
Fjende rammes i nærkamp	X					X
Fjende dør	X					

Hændelsestabellen bliver hurtigt mere kompleks og kræver typisk en kommentar. I dette tilfælde kan det for eksempel være nyttigt at fremhæve, at alle soldater kan rammes både af nærkamp og afstandsvåben, mens det kræver bestemte typer af soldater at bruge bestemte typer våben – en nøgledel af gameplayet. Kommentaren er ofte der, hvor der dukker en fornemmelse af et forløb frem.

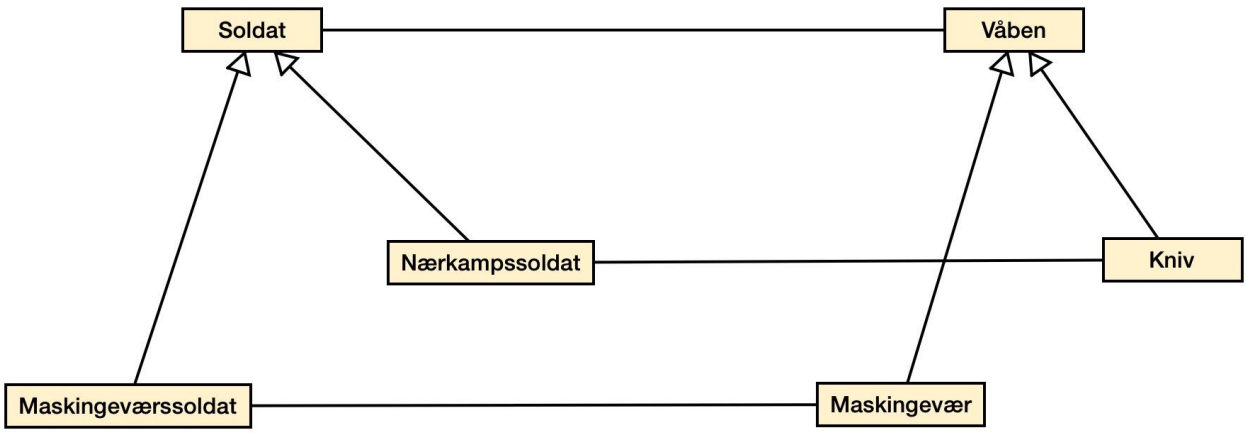
Hændelsestabellen bygger på de rige hændelses- og klasselister og på interaktionerne, der er beskrevet i use cases.

Samtidig peger hændelsestabellen fremad til klassediagrammet, hvor der skal noteres relationer imellem alle de klasser, der er involveret i fælles hændelser, enten i selve diagrammet eller forklaret i kommentarerne.

Klassediagram

Klassediagrammet viser, hvordan de forskellige ting i spillet hænger sammen. Det er det faste stillads, der danner rammen om hændelsernes dynamik. Der er to vigtige ting i forhold til dette. Dels hvilke klasser, der har relationer med hinanden, og dels hvilke generaliseringer, der gør sig gældende.

Relationerne mellem klasserne udspringer af, hvilke hændelser, der er fælles for flere klasser. Hvis vi tager vores sidste hændelsestabel ovenfor, kunne et klassediagram se sådan her ud:

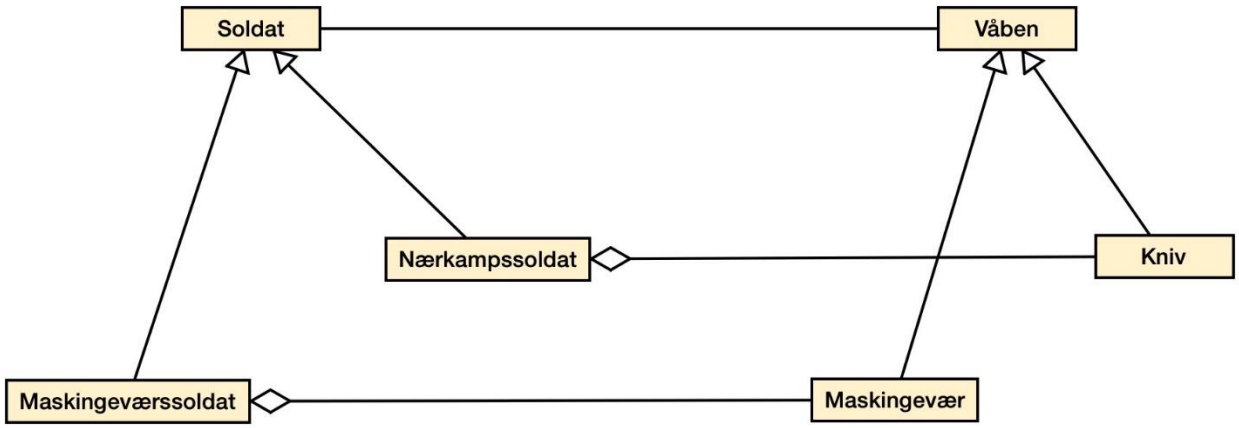


Pilen med det hvide hoved indikerer, at de klasser, som pilen kommer fra, er subklasser under den klasse, pilen peger på. Det vil sige, at de arver fra den. Der er nogle funktionaliteter, alle soldater

har, mens der er andre der ligger i deres specifikke våbentræning. Hvad disse funktionaliteter er, siger et analyseklassediagram ikke noget om, men det kan læses ud af hændelsestabellen.

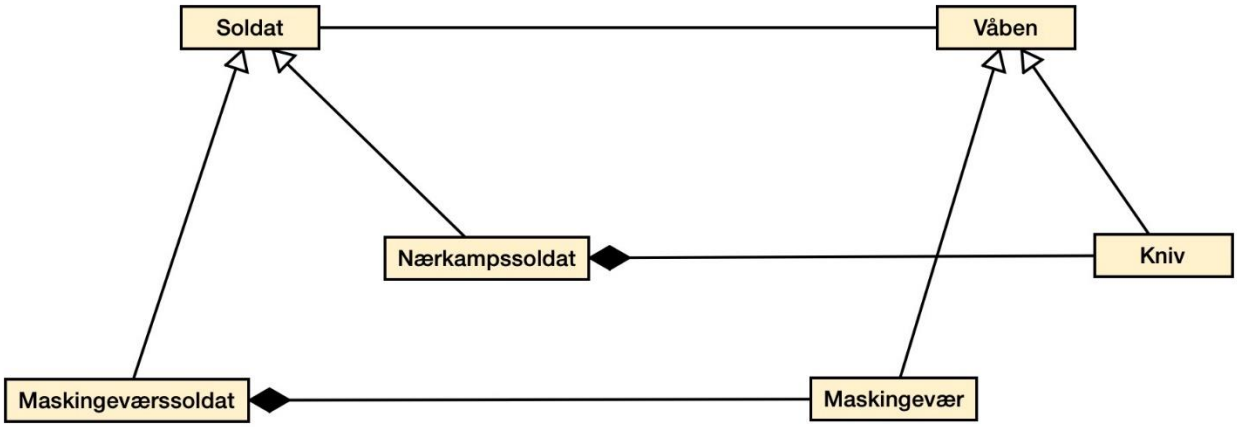
Ud over en generalisering og relation, er der to andre mulige forhold imellem klasser i et OOA-klassediagram. De er aggregeringer og kompositioner.

Aggregeringer er udtryk for, at en superklasse er udgjort af flere subklasser, der tilsammen udgør superklassen. Det klassiske eksempel er, at en bil er en superklasse, der består af subklasserne hjul, rat, motor og så videre. Man kan med en vis ret opfatte våbnet i et FPS-spil som en aggregering under de to soldat sub-klasser, således at våbnet altså er en del af soldaten. Det vil man notere således:



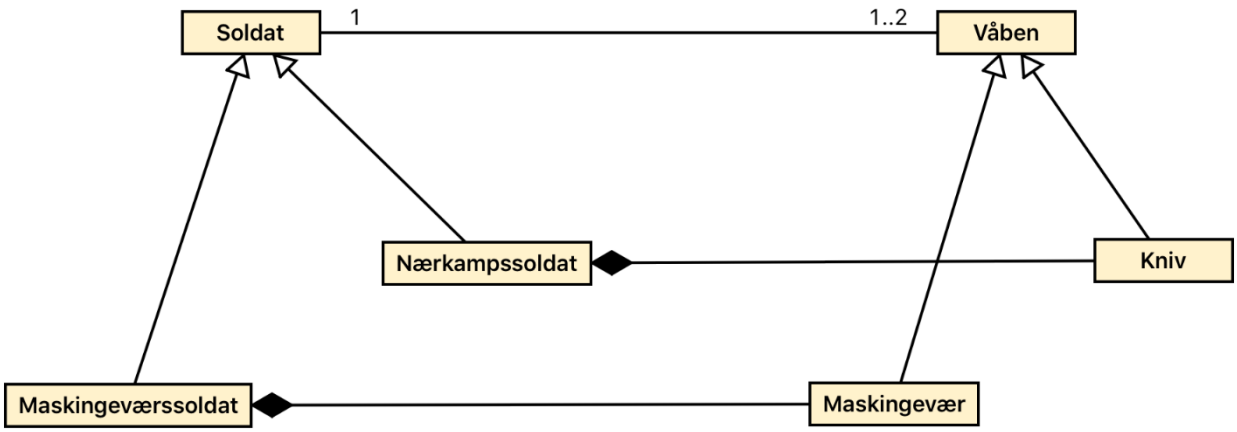
En aggregeret subklasse kan godt eksistere uden en instantieret superklasse. Det vil sige, at sådan som vi har noteret det ovenfor, vil et våben stadig kunne være til stede på skærmen, efter soldaten er død. Det er vigtigt, hvis en del af gameplayet er, at man skal kunne tage våben fra dem, man besejrer.

Der findes også et andet, aggregeringslignende forhold. Det hedder en komposition og viser et forhold, hvor hverken sub- eller superklasse kan overleve uden hinanden. Det er nyttigt, hvis vi vil lave et gameplay, hvor soldater starter med bestemte våben, for eksempel hvis de har forskellig våbentræning. Lad os sige, at soldaterne ud over forskellige plusser og minusser på grund af deres klasse også starter med forskellige våben, så nærkampssoldaten starter med en kniv, og maskingeværssoldaten starter med et maskingevær. Samtidig skal de forskellige soldater kunne samle våben op i banen, også selv om de ikke hører til deres specialevåben. Det noteres sådan her:

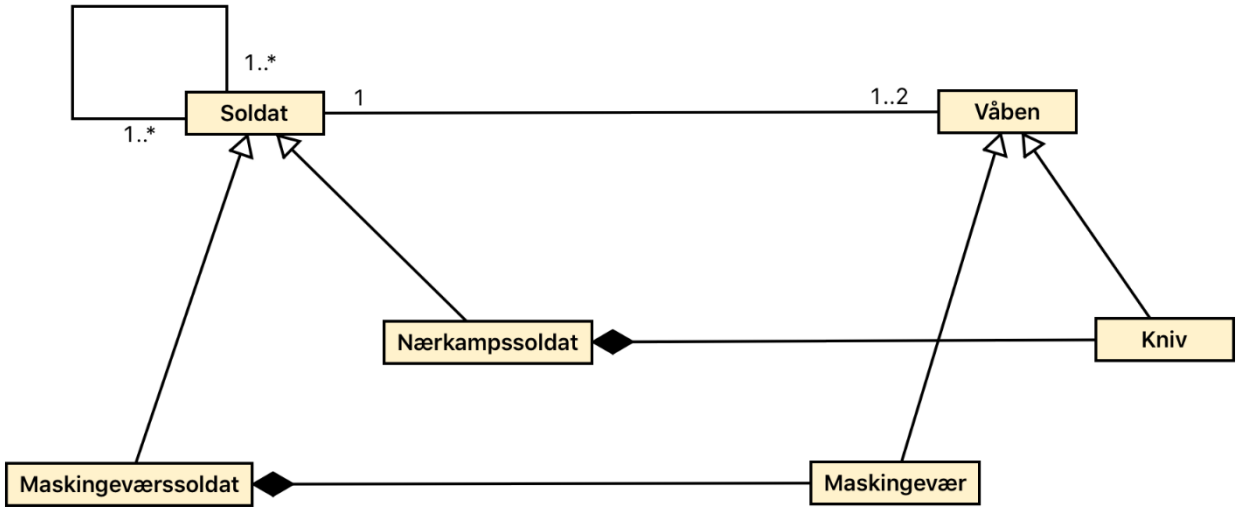


Vi markerer her, at hver gang der instantieres et objekt af klassens en nærkampssoldat, instatieres et objekt af klassen kniv også. Dette specifikke kniv-objekt forsvinder, når det nærkampssoldat-objekt, det er knyttet til forsvinder. Samtidig indikerer vi med generaliseringer, at objekter af begge superklassen våbens subklasser *kan* instantieres uden at være en del af en komposition – det er de våben, man kan samle op – og associationen mellem soldat og våben-klassen indikerer opsamlingssituationen. Som man kan se, er klassediagrammet ikke entydigt – det kunne tolkes på mange forskellige måder, sådan som det står oven for – men giver et holdepunkt for en samtale eller en skriftlig kommentar om spillets indretning.

Man bør også tilføje multiplicitet til sit klassediagram. Det vil sige, at man får et kort overblik over, hvor mange objekter af hver klasse, der er i spil, når systemet/spillet kører. Det noteres ved hjælp af symbolerne 0..1, 1, 0..*, 1..*, der betyder nul til et objekt, et objekt, nul til mange objekter, et til mange objekter. ”Mange”-symbolet * kan erstattes med et fast tal. Lad os sige, at man kan have op til 2 forskellige våben ad gangen i en kamp. Det ville vi notere sådan her:



Endelig mangler vi at gøre rede for én bestemt ting i klassediagrammet, nemlig at soldater kan interagere med hinanden – primært for at skyde hinanden, men dog stadig en central interaktion for vores gameplay. Det noterer vi ved det, der kaldes en selv-referentiel association, der markerer at objekter, der instantieres af den samme klasse kan have forhold til hinanden. Hvilket forhold, det er, kan man passende beskrive i kommentaren – og det bør kunne læses ud af hændelsestabellen. Det noteres sådan her, med lidt multiplicitet for at få detaljerne på plads:



Klassediagrammet trækker som nævnt på informationer fra hændelsestabellen.

Det peger fremad imod det mere detaljerede klassediagram i design-fasen, hvor der også integreres metoder, det vil sige kodelignende beskrivelser af, hvad klasserne kan, der skrives inde i klasserne.

Funktionsliste

Når først, det er afdækket, hvad der kan ske i spillet, er det nødvendigt at gøre sig det klart, hvordan hændelser og interaktioner skal understøttes af spillets funktionalitet. Det er her, vi identificerer, hvad spillet skal *kunne*, altså hvad det skal stille til rådighed for spilleren.

Dette gøres dels for at samle trådene fra de tidligere discipliner og dels for at prioritere arbejdsopgaver. Man identificerer simpelthen hvilken funktionalitet, der er vigtigst eller mest kompleks i et givet spil, sådan så man kan prioritere udviklingen af den funktionalitet, der er

vigtigst eller mest kompleks. Her er et eksempel på en funktionsliste med udgangspunkt i et fodboldmanagerspil:

Funktionsnavn	Prioritet	Involverede klasser	Involverede actors	Involverede use cases	Forslag til metodenavn
Vise holdopstilling	Høj	FootballPlayer, Team, Strategy	PlayerManager	Sæt hold, Vælg taktik	ShowTeam
Udregne og vise resultat af opgør	Høj	Team, FootballPlayer	PlayerManager	Spil kamp	PlayMatch
Vise transfervindue	Mellem	Market, FootballPlayer, Team	PlayerManager	Indkøb nye kræfter	ShowMarket
Vise sæsonplan	Mellem	SeasonPlan	Player	Planlæg sæson	ShowFixtures
Generere og vise random events	Lav	RandomEvent	Player	Modtag random events	GenerateEvent

Funktioner vil typisk blive oversat til metoder i design-fasen, og de danner i en spilsammenhæng grundlaget for gameplaybeskrivelsen i system definitionen.

Systemdefinition

Systemdefinitionen er konklusionen på den objektorienterede analyse. Den samler alle trådene i en sammenhængende tekstbeskrivelse af spillet, der kommer ind på områder som

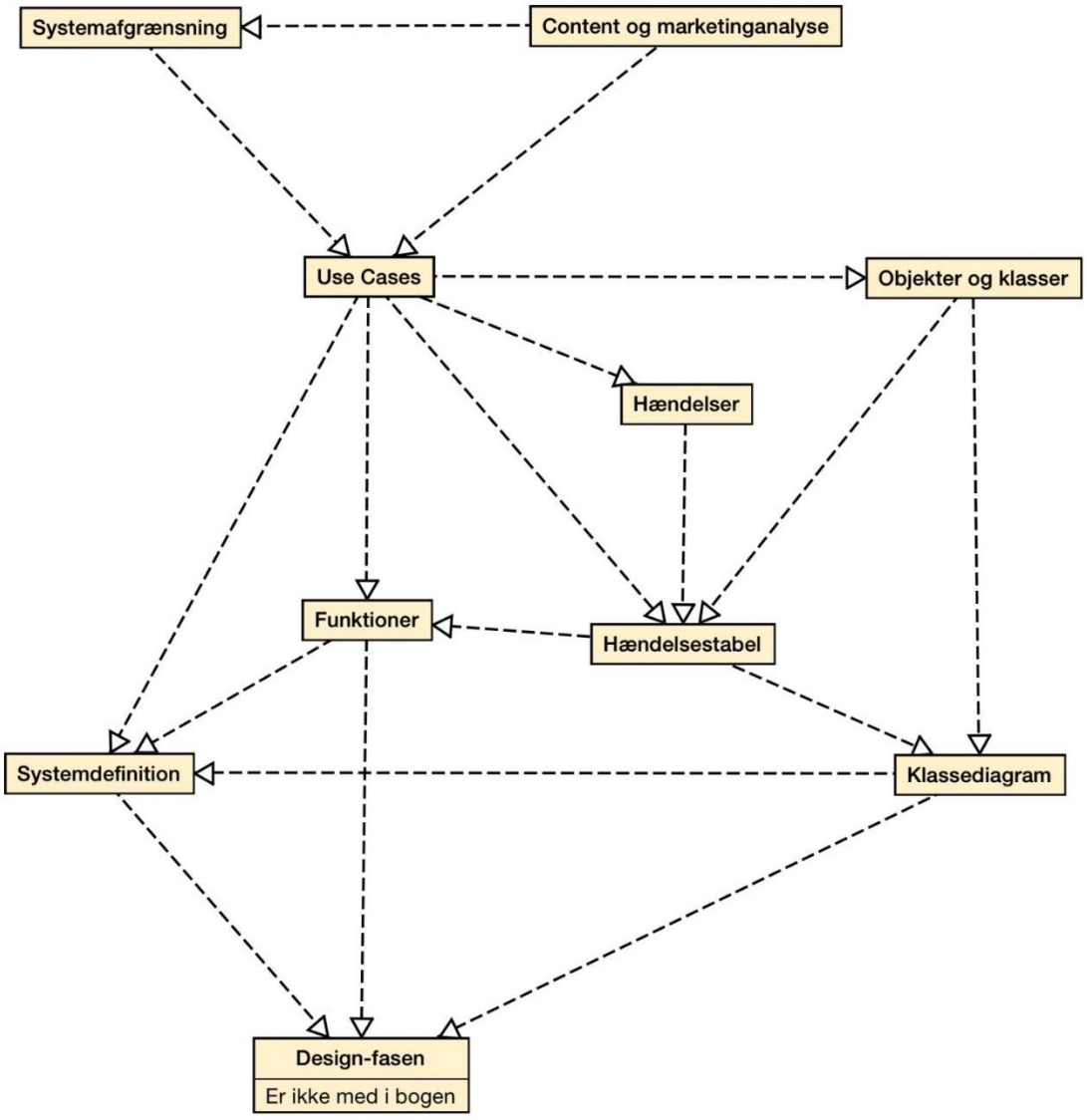
- Hvilken teknologi skal spillet udvikles med og til, herunder platformen spillet skal spilles på?
- Hvad er omstændighederne for udviklingen?
- Hvilke klasser og objekter er centrale?

- Hvilke funktioner skal understøttes?
- Hvilken rolle spiller spillet for brugerne? For kulturen som helhed?

Systemdefintionen skal kunne bruges som præsentation af systemet for en kunde, så fokus skal være på at præsentere, hvad analysen har afdækket af kundens behov og den mest effektive løsning på disse. Det kan derfor give god mening at sammenholde dele af system definitionen med marketingsanalysens konklusioner, og i øvrigt at have lægmandens synspunkt som udgangspunkt.

Gangen i OOA

Som det kan ses er en objektorienteret analyse en relativt kompliceret proces. I begyndelsen af kapitlet har vi forsøgt at opsummere processen ved at beskrive, hvor de forskellige dele kommer fra og hvad de fører til. Hvis man omsætter denne beskrivelse til et diagram, kunne det se sådan her ud:

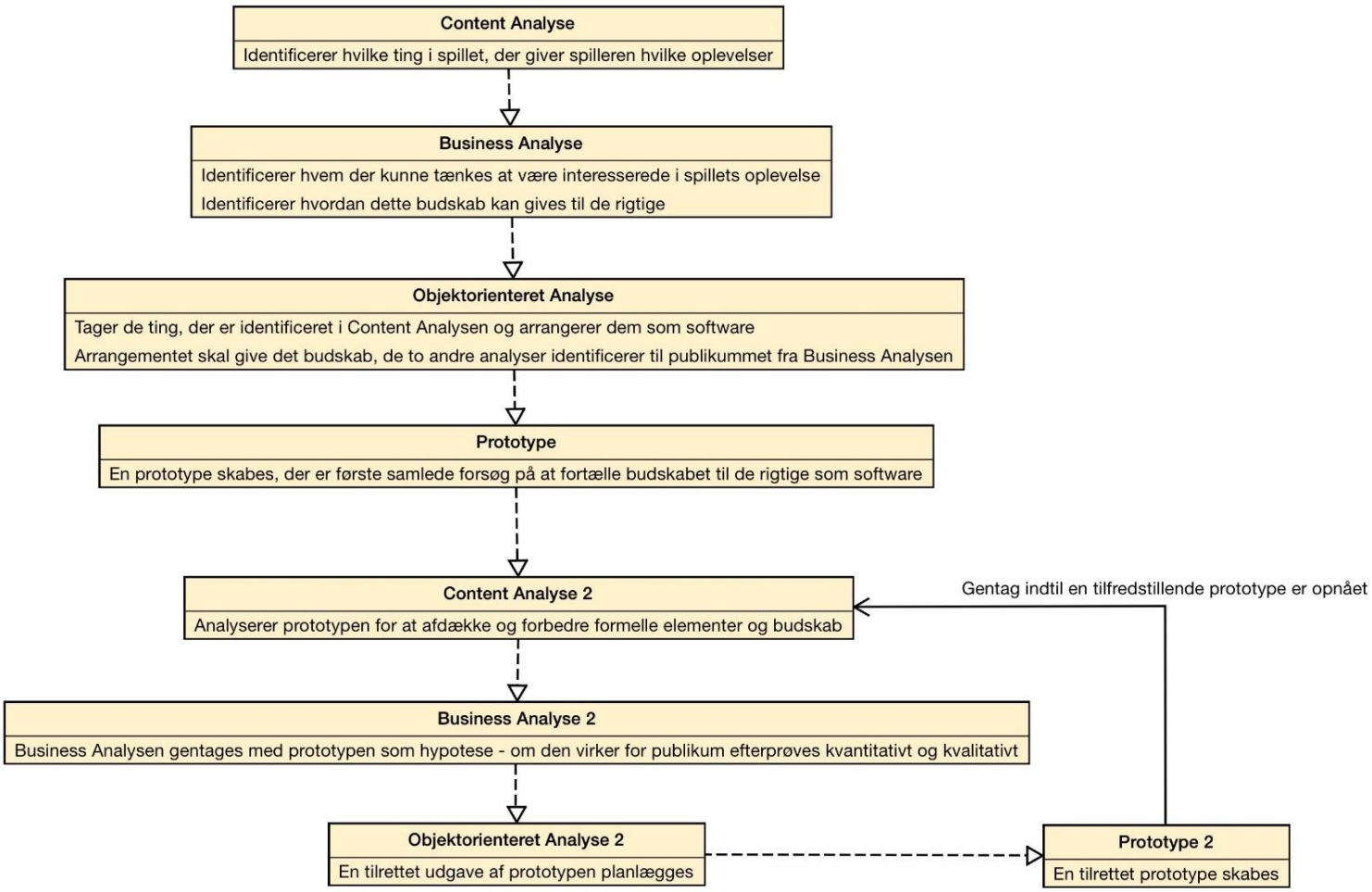


Som det kan ses er der en høj grad af sammenspil imellem de forskellige discipliner, og de ender alle med at give oplysninger videre først til hinanden og siden hen til den næste fase af spiludviklingen – designfasen. Denne fase ligger imidlertid uden for denne bogs ærinde.

SYSTEMATISK SPILANALYSE

Når man skal kæde en samlet systematisk spilanalyse sammen, er der flere forskellige måder at gribe det an på. Rækkefølgen af de tre overordnede analyseafdelinger ovenfor er valgt for at afspejle den fremgangsmåde, vi typisk vil råde til under opstarten af et spilprojekt.

Dette simple diagram skulle gerne kunne vise den ideelle sammenhæng:



Som det kan ses, bygger hver del af analysen ideelt set videre på de indsigter, der er kommet frem i de tidligere analyser. Som det også kan ses, har prototyper en vigtig plads i denne ideelle måde at praktisere systematisk spilanalyse. Imidlertid er prototyperne her set som et produkt af de forskellige analyser og ikke som et udgangspunkt. Der er store fordele ved at analysere inden den første prototype skabes, sådan som det er vist i de foregående afsnit.

At udvikle spil er typisk en iterativ proces. Det er svært at vide, hvad der er sjovt og hvad der fungerer, når man sidder og planlægger det, og derfor er det uundværligt at afprøve sine ideer.

Men det at få de ideer og i særdeleshed det at gøre det klart, hvordan disse ideer kan spille sammen til et budskab, der kan danne basis for en forretning og føres over i et stykke software – dét bliver væsentligt nemmere igennem en veludført systematisk spilanalyse.