## Indledning

***Beskriv i korte træk forskellene imellem objektorienteret udvikling, og komponentbaseret udvikling.***

For det første er det vigtigt at slå fast, at man ved begge udviklingsparadigmer arbejder med objekter. Forskellen mellem de to ligger i den måde man organiserer objekterne på. I klassisk OOP arbejder man med en klassestruktur, hvor man nedarver efter en X ***er en*** Y-kategorisering. Skulle man programmere vores spil fra bunden, og ud fra et OOP princip at Mojo ***er et*** GameObjekt, og Mojo ***er en*** bevægelig figur etc. Fordi det giver logisk mening.

Hvis man i stedet arbejder efter en komponentbaseret logik, så ville man sige at X ***har en*** Y-opførsel. Mojo ***har en*** Meshrenderer, og Mojo ***har en*** Rigidbody. En komponent kan således siges at være en *opførsel* som man kan tildele et objekt.

***Hvorfor benytter vi os af KBD i stedet for OOP?***

Fordi Unity er bygget op omkring komponenter.

Man kan godt designe en række underlæggende klasser der er bygget op efter traditionelle OOP principper, for så at lave et kodelag der implementerer MonoBehaviour, som er Unitys scripting api, ovenpå. Det kan være en god løsning hvis man laver et spil som skal lave tungere beregninger. F.eks. et rollespil eller et simulationsspil. Den slags er der ikke behov for i vores projekt, og derfor koncentrerer vi os fremadrettet kun om komponenter.

Unity kommer med en lang række forskellige komponenter indbygget, hvilket fritager os fra en lang række opgaver, vi behøver for eksempel ikke spekulere på at lave kollisionssystemer, eller renderer, fordi de allerede er lavet for os. Vores opgave er at levere de scripts, som i sig selv er komponenter, som skal forbinde GameObjekter og komponenter med hinanden.   
Der er en oplagt mulighed for at sikre lav kobling i forbindelse med udførelsen af disse, da et script uden videre kan bruges i et andet projekt senere hen. Så jo mere generisk vi kan udføre den opgave, jo større er muligheden for genbrug.

## Introduktion til brugergrænsefladen i Unity

Brugerfladen i Unity består af en række vinduer, der hver især tjener sit eget formål i forbindelse med udvikling. For overhovedet at kunne bruge Unity er det vigtigt at forstå formålet med, i hvert fald de følgende.

### Scene

Scene-vinduet er hvor man opstiller sin scene. Man kan positionere sit landskab, sine pro- og antagonister, kameraer, lydkilder og alle de andre dele som udgør spillet.

Et spil kan bestå af så mange scener som man ønsker, og det er helt og holdent op til udvikleren at afgøre hvor mange der er behov for. I vores prototype benytter vi en scene for hver menu, og en scene til vores spil.

### Game

I game-vinduet vises spillet som det ser ud, når det er renderet af scenens kameraer. Som sådan er det en rimelig approksimation af hvordan spillet ville tage sig ud hvis man lavede et build.

### Project

Under projekt kan man organisere sine assets, samt importere eller oprette nye. Project-vinduet er at sammenligne med en materialekasse, hvorfra man tager de redskaber som man skal bruge for at kunne sætte scenen op.

### Hierarchy

I hierarchy-vinduet kan man se hvilke GameObjekter man har tilføjet til den valgte scene, samt deres indbyrdes hierarki. I kombination med Inspector-vinduet øger bidrager oplysningerne her i væsentlig grad til at øge overskueligheden i scenen.

### Inspector

Inspektor-vinduet er kontekstuelt. Ved at markere et GameObjeck i hierarki-vinduet vises en liste over de komponenter som er tilknyttet dette GameObjekt. For hver komponent er har man både adgang til at manipulere den enkelte komponents public variabler. Og man kan tilføje et andet objekt til det valgte, for dermed at skabe en forbindelse imellem de to.

Hvis man derimod markere et asset under project-vinduet vises den type assets mulige importindstillinger i stedet i inspector-vinduet.

### Animator

Animator-vinduet er en del af Mecanim systemet. Fra dette vindue, kan man opbygge den tilstandsmaskine som håndterer under hvilke omstændigheder forskellige animationer skal afspilles.

## Spiludvikling med Unity

Hvad er de grundlæggende koncepter som vi arbejder med i unity?

Spiludvikling med Unity handler grundlæggende om at manipulere GameObjekter og deres komponenter. Tidligere i kapitlet beskrev vi hvordan komponentbaseret udvikling fungerede på et konceptuelt plan, herunder forklares det hvordan Unity helt konkret implementerer det.

### GameObject

Så snart man flytter et asset fra project, til scene (eller Hierarchy-vinduet), bliver der oprettet GameObjekt som kan indeholde det asset. I sig selv er et GO blot en beholder som alle typer assets og komponenter kan tilknyttes, for at opnå den ønskede funktionalitet.

Et GameObjekt består som minimum af et navn, et tag, et lag og en transform-komponent.

Det er også muligt at indlejre GameObjekter under hinanden, for at opnå et ønsket resultat. Et GameObjekt kunne være en mand, mens et andet er en hat. De to kan med fordel indlejres under et tredje GO kaldet: ”Mand-Med-Hat”.

### Component

En komponent er en ”opførsel” som man kan tildele et GameObjekt. En texture er en ”opførsel” der beskriver hvordan GO visuelt tager sig ud. Et script der får GO til at hoppe er en komponent osv.

Den eneste komponent som alle GameObjekter har tilknyttet, er en Transform. Denne komponent beskriver objektets position, rotation og skala i et tredimensionelt plan.

### Prefab

En prefab er at betragte som en plantegning over et givent objekt. Det er ofte en god ide, at lave en prefab ud af et GameObjekt når de ønskede komponenter er tilføjet og justeret. Man kan godt kopiere ”Mand-Med-Hat”-gameobjektet. Men det kan give problemer hvis man senere vil ændre i objektet. Tilføjede man f.eks. et par briller til ham, så ville kun det ene objekt blive opdateret. Hvis man i stedet lavede ”Mand-Med-Hat” til en prefab, så kunne man opdatere den, og så ville ændringen filtreres ud til alle kopier af denne prefab.

En prefab kan også instantieres på Run-Time via kode, hvilket er nyttigt hvis det ikke på forhånd er besluttet hvor og hvor mange af et givent GameObjekt der skal bruges i scenen.