# DOTS Cheat Sheet – Dansk

Skrevet af Daniel Kierkegaard Andersen

Indhold

[DOTS: 2](#_Toc64408323)

[ECS: 2](#_Toc64408324)

[Component: 2](#_Toc64408325)

[Normal playerloop: 2](#_Toc64408326)

[ArchTypes: 2](#_Toc64408327)

[Entities: 2](#_Toc64408328)

[Chunks: 2](#_Toc64408329)

[World: 2](#_Toc64408330)

[EntityManager: 3](#_Toc64408331)

[Systems: 3](#_Toc64408332)

[System Groups: 3](#_Toc64408333)

[Authoring Component: 3](#_Toc64408334)

[Convert-to-Entity: 3](#_Toc64408335)

[Sub-scenes: 4](#_Toc64408336)

[DynamicBuffers 4](#_Toc64408337)

[Jobs: 4](#_Toc64408338)

[EntityCommandBuffer: 4](#_Toc64408339)

## DOTS:

Data-oriented Technology Stack - Overnavnet for hele den nye teknologi

## ECS:

Entity Component System - Den tankegang der ligger bag DOTS hvor man splitter data op i data (components), og behaviour (systems)

## Component:

Gemmer data

## Normal playerloop:

Det normale playerloop ser ud som følgende, der hvor teksten er skrå betyder at det er en monobehavoiur event, her kan du altså se hvornår ting bliver kørt i rækkefølge.

Initialization>EarlyUpdate>\*FixedUpdate\*>PreUpdate>\*Update\*>PreLateUpdate>\*LateUpdate\*>PostLateUpdate

<https://medium.com/@5argon/1977de86e920>

## ArchTypes:

En kategori givet på et ens sæt af components - Altså to lister af componets der er hundret procent ens får samme ArchType.. Tilføjer man altså et component til en archtype ender man altså i en ny archtype

## Entities:

Instantieret ArchTypes, så hvor archtypes blot giver concepted af to components sammen, så kan entities altså holde data, tænk på det som prefabs vs gameobjects.. Prefabs er idéen om et objekt, mens gameobjects rent faktisk 'findes' og kan derfor holde data

## Chunks:

Dette er grunden til at ArchType findes, for hvis to entities har samme ArchType så bliver de gemt samme sted i dine ram, altså i samme \*chunk\*, dette gør det altså hurtigere at for ECS at lede gennem ting

## World:

Den verden som ens spil ligger I, der kan være flere men i vores tilfælde har vi kun en, ved navn World.DefaultObjectInjectionWorld

## EntityManager:

Holder alle ens entities i den World man er i, og holder dermed styr deres data.. og manipulationen af det

## Systems:

Tilføjer opførelse til ens data, så man kan ændre på entities component data, samt læse fra dem.. Disse bliver kørt såsnart at de er en klasse, og man behøver altså ikke tilføje dem til et game object

Det kan i nogen tilfælde være en god ide at sørge for at nævne hvornår de kører i forhold til hinanden, så det kan man gøre ved at skrive det her over sine system klasser:

[UpdateAfter(typeof(GroundCollisions))]

[UpdateBefore(typeof(Jumps))]

## System Groups:

En gruppe af systemer, forskellige grupper kan kører på forskelige intervaller. Men nogle af de vigtige grupper at kunne er SimulationSystemGroup, hvilke kører hvert frame, og FixedStepSimulationSystemGroup som kører hvert physicsupdate.. Ligger et system I en gruppe kører OnUpdate altså når den gruppe opdatere.. For at ligge et system i en gruppe kan man eksempelvis skrive følgende over sin system klasse:

[UpdateInGroup(typeof(FixedStepSimulationSystemGroup))]

## Authoring Component:

Et monobehvaiour component som tilføjer componentdata til ens entity

## Convert-to-Entity:

Omdanner game objects om til entities, samt kalder alle Convert methods på gameobjects der har monobehaviour scripts der implementere interfacet: IConvertGameObjectToEntity - Dette gør en istand til at lave ens egen authoring components, da man selv kan tilføje component data, et eksempel kunne være:

dstManager.AddComponentData(entity,

new GravityField {

GravityFieldType = gravType,

GravityAcceleration = gravityAcceleration,

Priority = priority

});

## Sub-scenes:

Gemmer alle dens gameobjects I en binær fil, som man kan loade og unloades. Derudover bliver alle objekter i en subscene automatisk tilføjet en 'convert-to-entity' gameobject component, som beder objektet blive genskabt men som entity

## DynamicBuffers

Arrays, men som components

## Jobs:

Hele ideen bag at kører systemer er så vi kan tage fuld advangtage af Unity's nye jobsystem, dette system er lavet så vi kan multi-thread kode uden at tænke over race-conditions, antal threads osv. Hvert job kan potientielt kører i sin egen thread, og man kan derfor kører massere jobs samtidig

## Structural Change

Når man skaber, eller fjerner entities, samt tilføjer eller fjerner components.

Dette kan ikke kørers i jobs/multithreaded kode.

## EntityCommandBuffer:

En buffer der gemmer en liste af kommandoer, som bliver executed i rækkefølge når anmodet.

Smart når man ønsker at lave ændringer til ting der skaber structural change, da det er den eneste måde at gøre det.