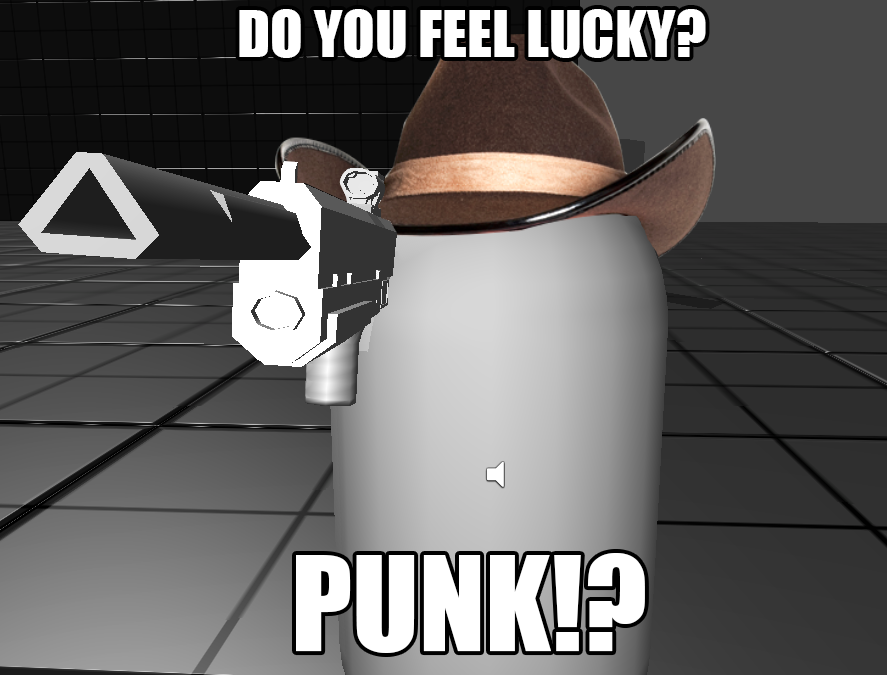
Capsule Combat 2:  
Revenge of the capsules



**Projektgruppe:**  
Asger Gitz-Johansen  
Thomas GwynfrynMcCollin

Indhold

[Abstract 3](#_Toc386489374)

[Problemformulering 4](#_Toc386489375)

[Funktionsbeskrivelse 5](#_Toc386489376)

[Layout 5](#_Toc386489377)

[Use Case 5](#_Toc386489378)

[Programdokumentation 6](#_Toc386489379)

[Overordnet programbeskrivelse 6](#_Toc386489380)

[Detaljeret gennemgang af udvalgte programdele 6](#_Toc386489381)

[Blok og FLOW diagrammer 9](#_Toc386489382)

[Konklusion 10](#_Toc386489383)

# Abstract

I've been assigned the task of designing and creating an online competitive first person shooter video game, where you can either team up or go against your friends in an arena type action. This report reflects my thoughts on how the programming of my product and process have been in this assignment. I will explain in detail how I dealt with the challenge of spawning and de-spawning players that die and gets revived to join the game/slaughter again.

# Problemformulering

Jeg skal lave et produkt til en kunde (læreren). Produktet skal være et spil, som centralt består af en multiplayer komponent hvorved 2 til x personer kan løbe rundt i en arena og skyde hinanden ihjel. Udover det viser research at spillere, som ofte spiller den slags spil også kan lide at kunne lave deres egne våben. Dette er derfor også et krav til produktet.

# Funktionsbeskrivelse

## Layout

Programmet er et online multiplayer kompetitivt skydespil med fokus på at slå andre spillere ihjæl. Det visuelle layout af spillet er meget inspireret fra andre skydespil såsom Call of Duty, Halo og Battlefield. Eftersom spillet er et skydespil gav det kun mening at sætte geværet i det nedre højre hjørne. Også da de fleste folk er højrehåndet, giver det også mening for spilleren at "holde" våbnet i højre hånd på skærmen.   
Den grafiske stil i spillet har jeg valgt skulle være en form for neon-cyber-sci-fi stil med meget fremtrædene farver brugt allermest til highligts af vigtige objekter i spillet.

## Use Case

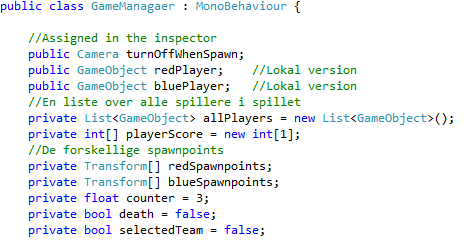
En gruppe brugere sætter spillet op til et LAN-party. En fra LANet vælger at hoste et spil og ved hjælp af enten en ekstern hjemmeside eller ved at åbne en komandopromt, finder han ud af hvad hans ip-adresse er. Han fortæller de andre til LAN-partyet hans addresse, så de kan logge ind på den. Så snart der er en der er forbundet til serveren bliver scenen kaldet "Map01" loadet og hver spiller får muligheden for at vælge et hold de gerne vil spille som: Rød eller blå Hvert hold spiller da mod hinanden og målet er at skyde så mange modstandere som muligt. Spillerne har også mulighed for at samle en skjold power-up op, som giver dem en lidt bedre fordel end de andre spillere. Hver spiller har også mulighed for at kunne customize sit våben. Ved at trykke på tasten "T" toggler man customization menuen, som man så kan bruge til at ændre sit våben. Det har dog ingen effekt på gameplayet at ændre på våbnet.  
Det var planlagt at når en spiller døde ville der være et scoring system, som ville holde styr på de to holds point og der ville være udvalgt en vinder af kampen når en hvis mængde point var opnået. Denne feature blev dog forhindret eftersom vi løb tør for tid.

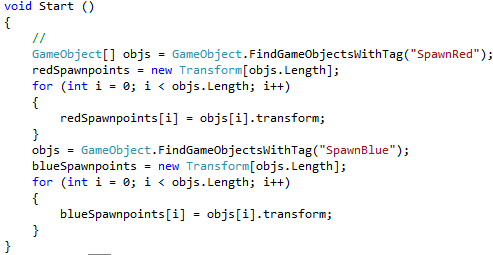
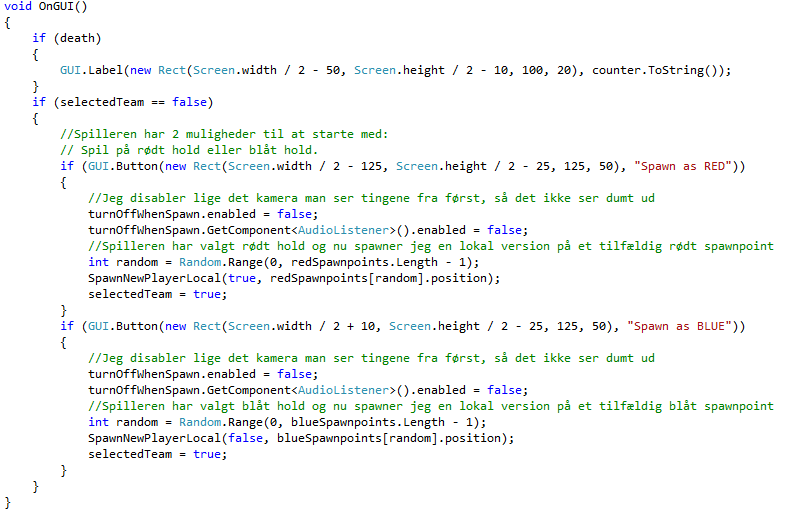
# Programdokumentation

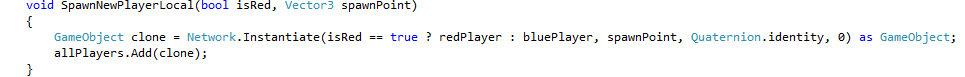
## Overordnet programbeskrivelse

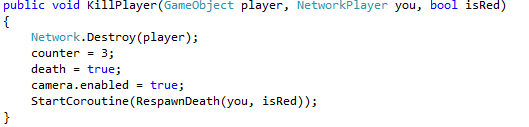
Jeg har valgt at anvende unity3Ds indbyggede RPC calls til at sende næsten alt information frem og tilbage mellem klienter. Alternativet var at bruge unity3Ds NetworkView komponents state syncronization, hvilket jeg personligt føler er meget buggy og har tendens til at glitche helt vildt ud når man anvender mange af dem, som jeg gør i Capsule Combat 2. Hver gang man skyder, spawner man en effekt over netværket, som sletter sig igen rimelig hurtigt efter den er blevet spawnet. Hvis jeg brugte state syncronization til at synkronisere positionen og rotationen af effekten i stedet for et enkelt call når de spawner, ville spillet glitche ud oftere end det allerede gør.

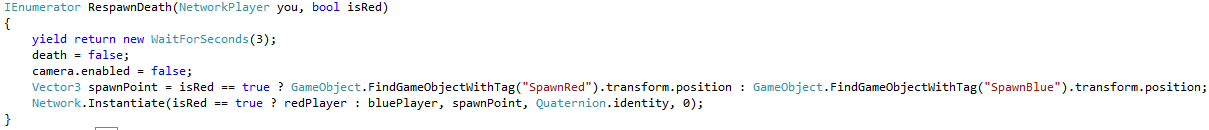
## Detaljeret gennemgang af udvalgte programdele

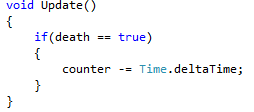
Jeg har valgt at beskrive GameManagaer.cs eftersom den indeholder noget interesant netværks kode. Jeg vil derfor gå i dybde og næsten linje for linje forklare hvad der foregår. Kildekoden følger med arkivet, så hvis noget af koden her i journalen virker lille eller ulæselig på nogen måde, råder jeg at man åbner koden ved siden af og følger med.  
  
I starten af min klasse GameManager som nedaver fra MonoBehavior har jeg defineret mine variabler: turnOffWhenSpawn er af typen Camera og er sat til at være det kamera der er aktiveret, når man er død lokalt. Det kamera er også aktivt inden man vælger et hold, og jeg deaktiverer det længere ned i koden når man spawner på et hold. redPlayer og bluePlayer er sat til at være deres respektable prefabs så jeg kan bruge dem til at spawne spillerne. allPlayers er en liste over alle spillere som er forbundet til spillet. Den er dog ikke rigtig i brug eftersom jeg ikke fik lavet scoring. Det samme gælder for variablen playerScore. redSpawnpoints og blueSpawnpoints er lister af de forskellige spawn punkter der er på mappen. Jeg har kun et spawnpoint for hvert hold i den nuværende bane, men det er ment at der skal være flere. Variablen counter bruges til at nedtælle hvor lang tid der er tilbage når man er død. Den bruges kun til den visuelle del og ikke den faktiske nedtælling. Variablen death er hvad den lyder som. Når den er af falsk værdi er den lokale spiller i live, hvis ikke er han død. Den sidste variabel, som jeg definerer i starten af klassen er selectedTeam. Den bruges til at tjekke om man har valgt sit hold og bruges i sammenhæng med turnOffWhenSpawn variablen.

  
I metoden Start() sætter jeg variablerne redSpawnpoints og blueSpawnpoints til deres respektive spawnpoints. Dette gør jeg ved at først finde alle objekter med tagget "SpawnRed" og giver redSpawnpoints de værdier og derefter finder jeg alle objekter med tagget "SawnBlue" og sætter blueSpawnpoints til at være lig det. Til at tildele værdierne anvender jeg for løkker.  
  
I metoden OnGUI(), som bliver kaldet en gang hvert event[[1]](#footnote-2) har jeg hovedsagligt 3 ting der sker. I starten tjekker jeg om den lokale spiller er død. Hvis han er det bliver der vist en GUI.Label som viser hvor lang tid der er indtil han spawner igen. Derefter tjekker jeg om man ikke allerede har valgt et hold, hvis man ikke har det bliver der vist to GUIknapper: en med teksten "Spawn as RED" og en med "Spawn as BLUE". De gør stort set det samme begge to udover de spawner deres respektive farves prefab på deres respektive farves tilfældige spawnpoint.

  
Metoden SpawnNewPlayerLocal() burde egentlig hedde SpawnNewPlayer eftersom min metode SpawnNewPlayerClone, som var ment til at spawne spilleren på de andre klienter endte med ikke at blive brugt. Det som SpawnNewPlayerLocal() gør er at spawne en instans af enten en rød spiller eller blå spiller på det angivne spawnpoint. Både hold værdi og spawnpoint er to parametre som metoden har og er krævet at udfylde når man kalder metoden.

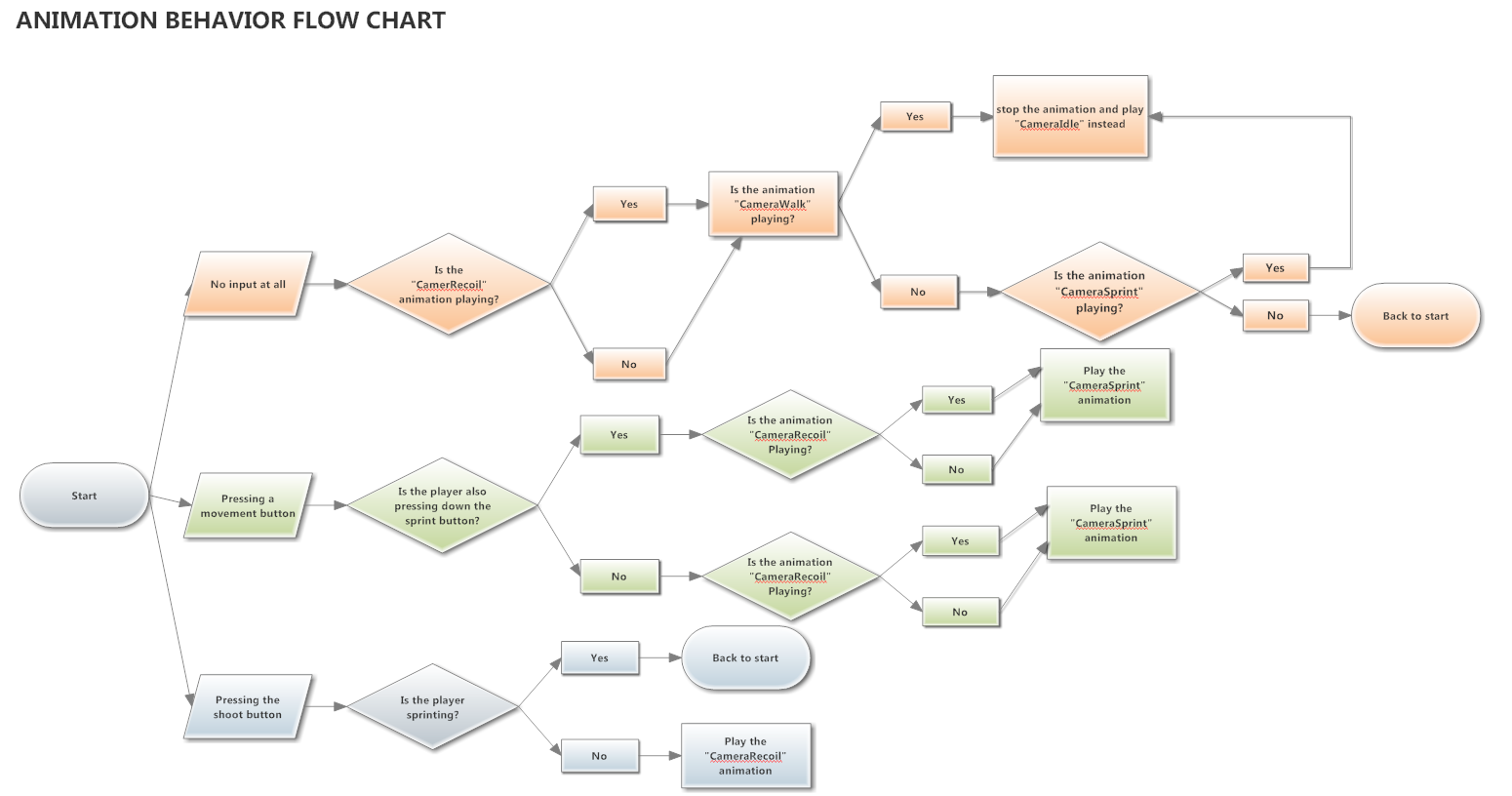
  
Metoden KillPlayer() har 3 parametre: player af typen GameObject, you af typen NetworkPlayer og isRed af typen boolean. Metoden starter med at slette refferencen player over netværket og begynder derefter at tælle ned fra 3 (respawn timer) ved at starte metoden RespawnDeath().

  
Metoden RespawnDeath() er af typen IEnumerator og kan derfor bruges til at vente en bestemt mængde tid ved at returnere en WaitForSeconds. Det gør jeg også i denne metode. Jeg får den til at vente præcis 3 sekunder og derefter sætter den death til at være falsk, deaktiverer kameraret som nævnt oppe i variabel definitionen og spawner en ny spiller på et spawnpoint.



Til allersidst har vi metoden Update(), som bliver kaldt en gang hvert frame af Monobehavior. Det eneste jeg gør inde i den er at tælle counter ned, når man er død.

# Blok og FLOW diagrammer



Figur 1  
Kameraet har en masse forskellige animations-stadier, som den kan være i. Jeg brugte dette FLOW chart til at finde rundt i hvilke animationer skulle afspilles hvornår, hvordan og hvorfor. (en lidt større version ligger sammen i .zip mappen)

# Konklusion

Jeg har produceret et spil, hvor man kan forbinde til en host, eller være en host. Skyde hinanden i en arena agtig stil. Man kan customize sit våben og man kan respawne. Der er endda blevet tænkt over at tilføje variation til gameplayet med skjold mekanikken. Så derfor kan jeg konkludere at jeg har udført alle kravene der opholdte sig i opgavebeskrivelsen.

1. http://docs.unity3d.com/Documentation/ScriptReference/Event.html Forklaring af unitys Event [↑](#footnote-ref-2)