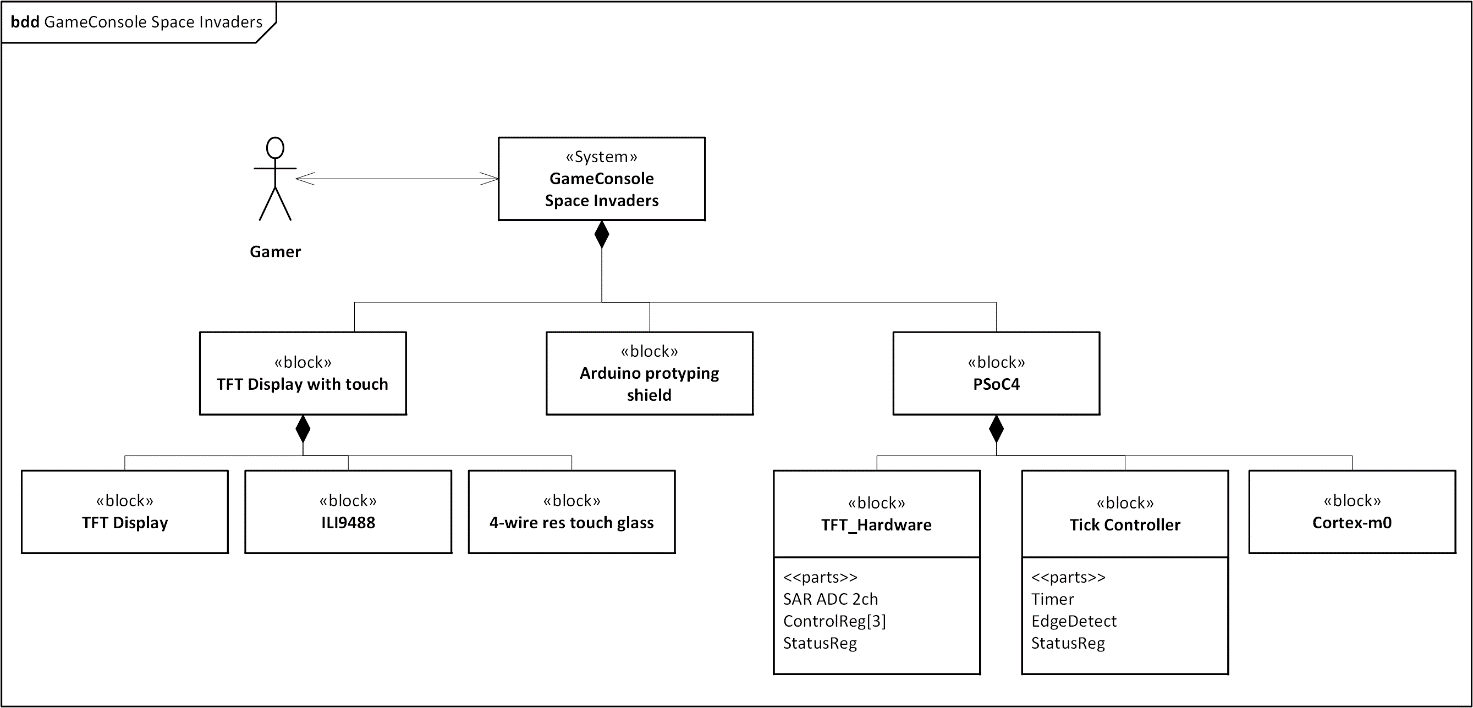
# System Description (BDD, Sekvens etc)

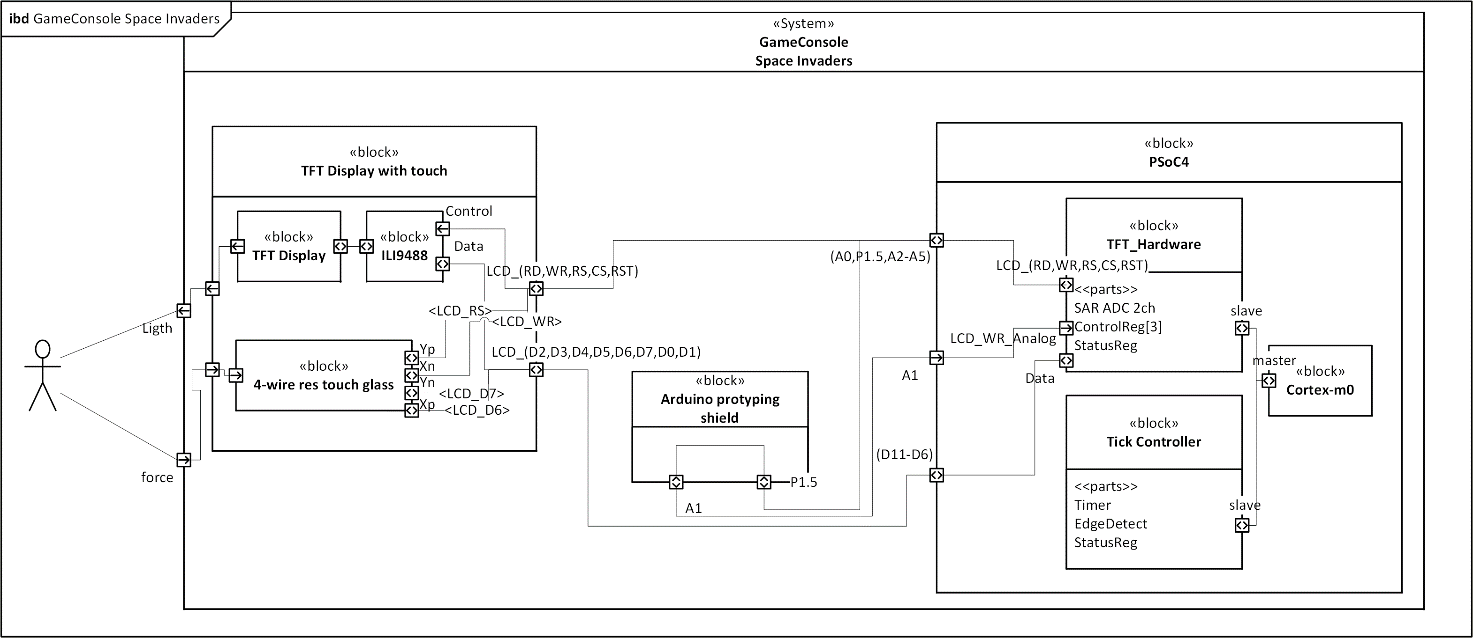
## Blok diagrammer over Space Invader

### BDD



Figur 1 - bdd GameConsole Space Invaders – Game systemet består af tre hoveddele, et TFT Display der er beregnet til Arduino uno; et Ardiono protyping shield og en PSoC4. – TFT Display with touch: Er systemets interface, det bliver brugt af brugeren til at interagere med og systemet bruger displayet til at vise spillet til brugeren. Displayet består af et TFT display der bliver styret af en ILI9488 skærm kontroller, med indbygget grafisk hukommelse. Touch delen består af et 4-wire resistiv touch glass system, som sidder ovenpå skærmen og ændre sine resistive værdier udfra hvor brugeren trykker, og hvor hårdt. – Arduino protyping shield: Er et ekstra komponent, der var nødvendig at tilføje på grund af pin begrænsning i PSoC4 arkitekturen, bruges til at tilkoble to pin sammen på PSoC4 (ekstern rute); En dybere forklaring kommer i Topdesign beskrivelsen. – PsoC4: Består af en lang række komponenter, eller ret og sagt en masse digtale blokke og analoge blokke og en enkel Cortex-m0. PSoC4 giver en mulighed for selv at vælge hardware opbygningen. TFT\_Hardware: Der er i dette projekt bygget noget internt hardware til at integrere med ILI9488 enheden og 4-wire res touch glass. Til dette er der brugt en SAR ADC, tre Control register og et Status register. Tick Controller: er noget hardware der bruges til at styre noget software på Cortex-m0, bliver uddybet i Topdesign beskrivelse.

### IBD



Figur 2 - ibd GameConsole Space Invaders – Dette giver et overblik over hvordan blokkende i systemet er koblet sammen.

# System architecture and user interface

## Tekniske overvejelser – individuelt

## Fordele og ulemper ved forskellige implementeringer – individuelt

## Topdesign indepth (guldkorn)

### EMULATED EEPROM – Jonathan

Der er i produktet brug for noget non-volatile hukommelse, til at gemme High score. Dog var der den ulempe at PSoC4 ikke har en indbygget EEPROM, som ellers ville have været optimale valg.

For at løse dette problem har Cypress lavet en emulator, der giver mulighed for at tilskrive til flash området, hvor koden ligger. Dette område er som standart kun et ”read only” område, og PSoC4 skal være i en rigtig tilstand for at tilskrive til flash området.

Eksampel: udsnit fra “GameEngine.c”

static const uint8 CYCODE HighScore**[**4**]** **=** **{**0**};** // fortæl at dette skal gemmes på flash ved at bruge postfix CYCODE der fortæller hvor variablen skal gemmes.

**...**

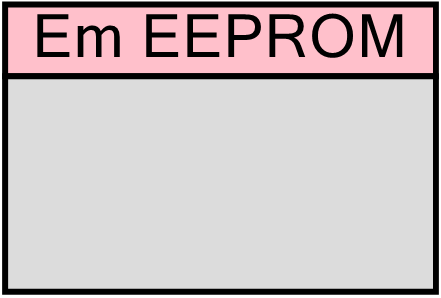
uint8 newHighScore**[**4**];**

uint32 HighScore\_ **=** **(\*(**volatile uint8 **\*)&**HighScore**[**0**]);** // aflæsning fra flash område.

**...**

status **=** EEPROM\_1\_Write**(**newHighScore**,** HighScore**,** 4**);** // Hjælpe funktion fra Cypress der sørger for at tilskrivning forgår korrekt.

For at få adgang til funktionen skal man først inkludere EM\_EEPROM i topdesignet, som er en ren software blok, der indeholder en brugbar funktion (de andre er tomme) ”EEPROM\_1\_Write”.

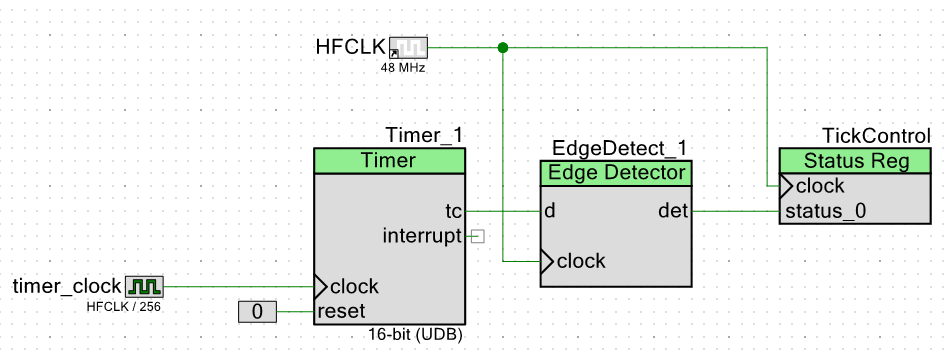


Figur 3 - Cypress EMULATED EEPROM, som tilføjer et API til at kunne sikret tilskrive til FLASH området hvor Koden ligger.

### TICK Controller – Jonathan

På grund af den måde der er valgt at bygge spillet på, er der påkrævet at spillet skal have et fixet tick interval, for at sikre at spillet ikke lige pludseligt køre hurtigere eller langsommer fra tick til tick.

“*Nearly all video games (including Minecraft) are driven by one big program loop. Just as every gear in a clock is synchronized with the pendulum, every task involved in advancing a game simulation is synchronized with the game loop. Appropriately, one cycle of the game loop is called a* ***tick*** “[[1]](#footnote-1)

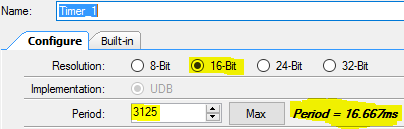


Figur 4 – Hardware opsætning af TICK kontrolleren. Der er brugt 4 komponenter til at bygge TICK Kontrolleren. En Timer blok der er sat op til at generere et trig hver gang den rammer en ønsket værdi, hvorefter den vil resette sig selv til 0 igen og tælle op igen. Til at trigge Timer’en er der brugt en nedskaleret clock af HFCLK. På grund af at der er brugt en nedskalleret clock af HFCLK, behøves der ikke noget Hardware til at synkronisere signalerne. Men fordi vi kun er interesseret i trigget skal der bruges en EdgeDetect komponent, for ellers vil det ligne at den generere en masse trig’s. Som det sidste trin er der brugt et Status register, der er sat op til at være et sticky register (Vil blive holdt høj, hver gang der kommer et trig, indtil en aflæsning sker, hvor den så bliver kørt lav igen).

Opsætning:

Timer clock: grunden til at der er brugt en nedskalleret clock af HFCLK, skyldes at der ikke kunne implementeres en stor nok timer, på grund af mangel på UDB’er[[2]](#footnote-2).

Timer\_1: Timeren er sat up til at generere et trig(tc[[3]](#footnote-3) signalet går høj) hver gang dens værdi er lig med 0. I dette design er den sat til at generere et trig hvert 16.67ms, som svare til 60 trig i sekundet.



EdgeDetect\_1: Bruges til at detektere hver gang tc signalet går højt.

Status Reg (TickControl): Et sticky register som der kan aflæses fra. Når det bliver sat højt vil det holde værdi indtil der sker en aflæsning.

Det eneste der kræver for at få det hele til køre over i software domænet, er at initialisere Timeren. Og efter hvert tick forløb, vente på at TickControl er lig 1 igen.

Udklip af kode implementation af TICK Controller på softwaresiden, udklippet er fra ”SpaceInvader.c”

**while(**1**)**

**{**

**...** //game code

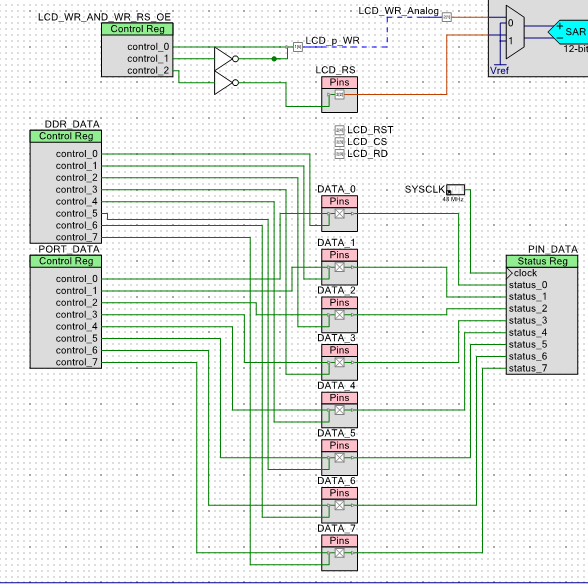
**while(**TickControl\_Read**()** **==** 0**);**

**}**

Denne løsning har et par ekstra fordele, f.eks. hvis der er et tick der tager for lang tid, (f.eks. hvis mange skyd skal allokeres) vil den komme ud af vente løkken med det samme, og få mulighed for at indhente næste tick. Selvfølgelig er der mulighed for at hvis tick hastigheden er for høj til at spillet kan følge med, og derfor ikke vil opfange alle Trigs fra TICK Controlleren. Dette er selvfølgelig noget der skal indstilles lidt på, for at få den helt rigtige værdi. (Dette design køre med 60 ticks i sekundet).

### TFT\_Control - Register control – Jonathan

TFT Skærmen der blev brugt har pin layout til en Arduino uno, som også passer til den enhed der bliver brugt i dette projekt: psoc 4 pioneer kit[[4]](#footnote-4). Dog med den lille ulempe at port layoutet ikke var ens, og ville koste mange software operationer bare at skifte værdi på data pins. For at løse dette problem, blev der brugt et par Control register og et Status register, til at lave en løsning der ikke ville koste for mange clock perioder. Man kan godt sige at portene er blevet omplaceret, så det passer med Arduino uno.



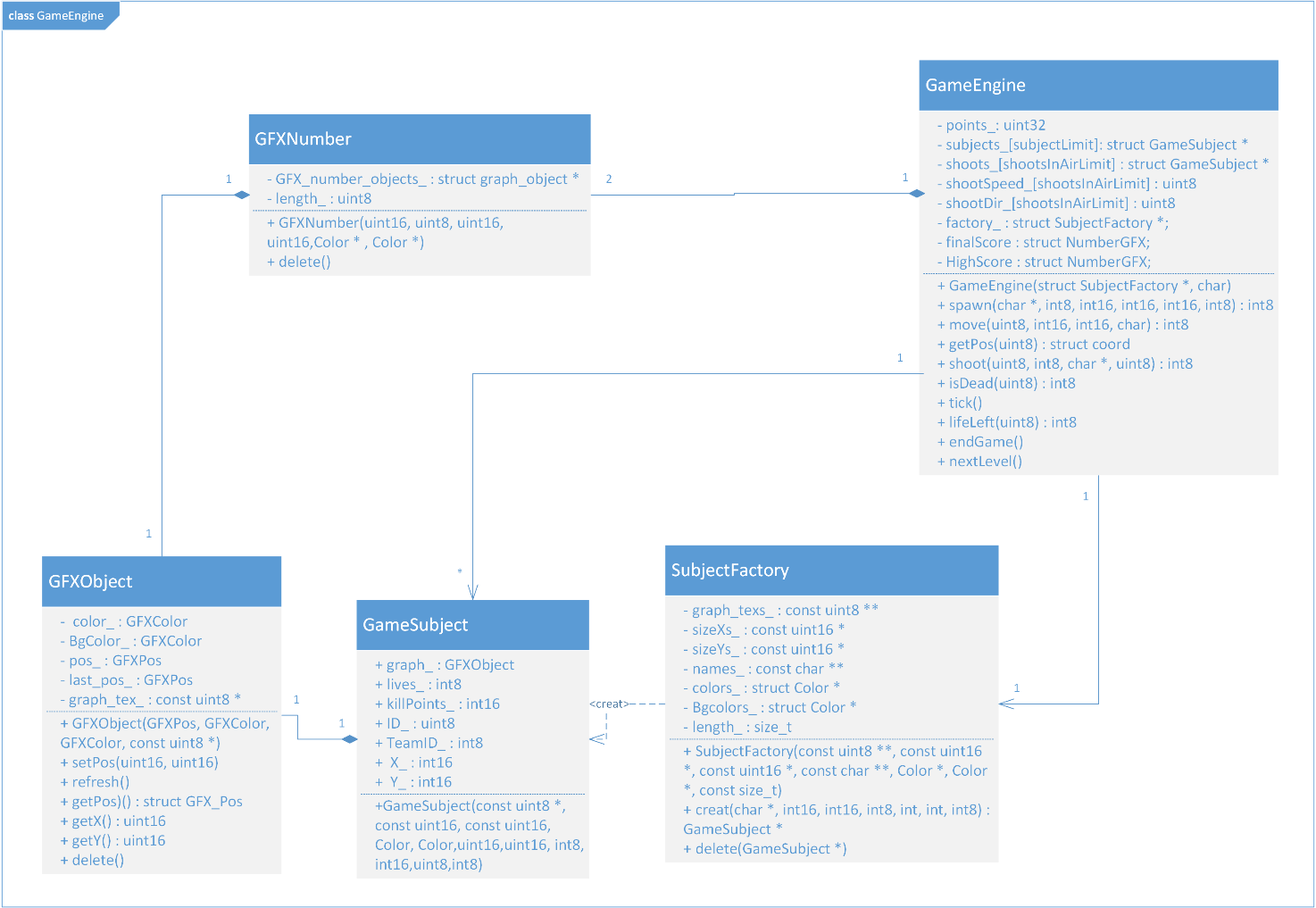
Figur 5 – Til DATA bene er der brugt to Control register ”DDR\_DATA” og ”PORT\_DATA”, og et Status register PIN\_DATA. DDR bruges til at aktiver output, PORT bruges til at tilskrive værdier til bene, PIN kan aflæse fra bene. Der er bygget et lidt speciel Control register ”LCD\_WR\_AND\_WR\_RS\_OE”; control\_0 bruges til at bestemme output værdien for LCD\_WR; control\_1 bruges til at aktivere eller deaktivere outputtet for LCD\_WR, note er aktiv lav; control\_2 bruges til at aktivere outputtet for LCD\_RS, note aktiv lav. Grunden til at OE er aktiv lav skyldes at TFT Displayet allerede var implementeret før dette Control register blev lavet, for at sikre at der ikke skulle opstå uventet resultater efter Implementation, da ”LCD\_WR\_AND\_WR\_RS\_OE”, kunne bruges som bare et Control register for ”LCD\_WR”. ”LCD\_p\_WR” og ”LCD\_WR\_Analog” er vores ”LCD\_WR”. ”LCD\_p\_WR”: ”p’et” er en husker på at den skal styres fra ”LCD\_WR\_AND\_WR\_RS\_OE”; ”LCD\_WR\_Analog” er en analog pin der er loddet til ”LCD\_p\_WR” så det er muligt at måle spændingen (bruges i touch delen.).

”LCD\_p\_WR” og ”LCD\_WR\_Analog” kan virke som en lidt mærkelig løsning, især når man kigger på ”LCD\_RS” hvor det hele er samlet i et ben. Grunden til dette skyldes PSoC4 arkitekturen. Man kan ikke have en hardware digitaloutput og OE sammentid med et analog input ben. Selvfølgelig kunne dette løses ved at undlade at have den hardware digitaloutput; Dette vil koste omkring 6 ekstra clock perioder på grund af shift- og and/or- operationer (og lidt mere). Dette ser ikke ud af meget, men eftersom dette ben skal toggels 4 gange for hver pixel tilskrivning, runder det hurtigt op og gør koden meget langsom. Hvor denne løsning kun kræver en enkel register tilskrivning, som er en meget hurtigere løsning.

## Klassediagrammer

## GameEngine

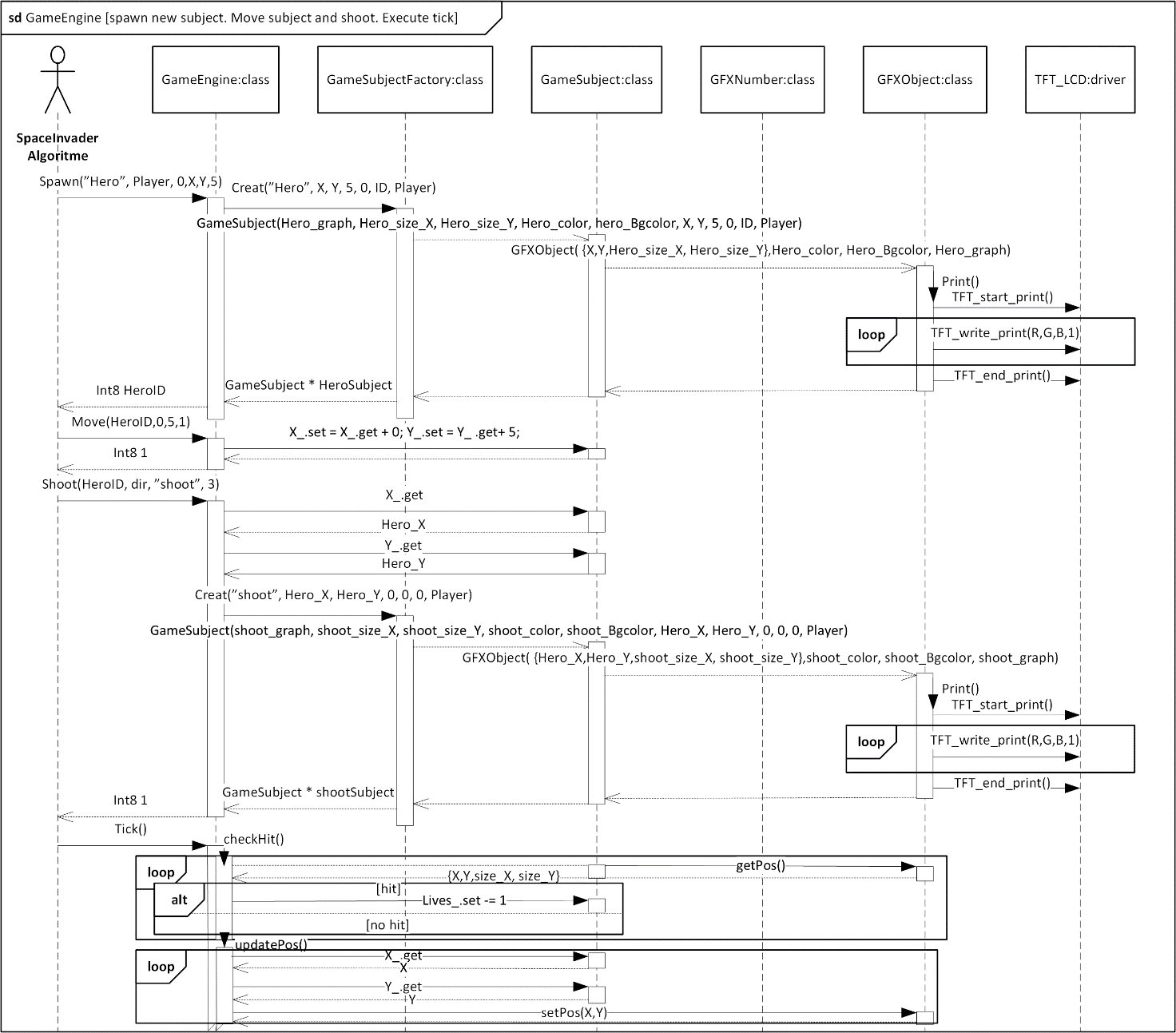
GameEngine spiller en central rolle i produktet, og står for håndtering af subjecter og giver udvikleren mulighed for at bruge et par basale funktioner på dem, som at flytte dem, tjekke deres status (så som liv), eller affyre et skud af, som GameEngine også håndtere. GameEngine gemmen løber ikke alle Subjecter hver gang en metode bliver kaldt. F.eks. hvis der skal flyttes på 16 Subjecter før det næste tick, vil de ikke blive opdateret løbende, men først når metoden tick bliver kaldt i GameEngine, hvor den vil opdatere nye positioner, skyd simulering og check om der er et subject der er ramt af et farligt skyd (et skyd med et andet hold nummer end det nummer subjectet har.)



Figur 6 -Klassediagram GameEngine. – GameEngine er den klasse der står for håndtering af GameSubject’s og styre dem udfra de kommandoer der ønskes hvert tick. Måden man styre et GameSubject på, er via det unikke ID man får returneret når spawn metoden bliver brugt. Med ID’et kan man flytte GameSubjectet, udføre handlinger (kun skyd er implementeret) eller tjekke hvor mange liv GameSubjectet har, eller om det er dødt. Den måde GameEnige spawner et nyt GameSubject på, er ved at bruge GameSubjectFactory til allokere og oprette og initialiser et GameSubject med metoden ”creat” hvor den så returner en pointer til GameSubject’et, hvis den er løbet tør for hukommelse returner den en NULL pointer, som vil medføre at spawn kommandoen vil returner -1. GameSubject Indeholder en række oplysninger, koordinater (den position den vil have næste tick), ID (et unik ID for Subjectet), TeamID (Hold nummer, bruges til at undgå at spaceinvaders skyder hinanden), killPoints (hvor mange points man får når et GameSubject bliver drabt), lives (Antallet af liv et GameSubject har tilbage), graph (Er det grafiske object der repræsentere subjectet). GFXObject er et grafiske object og står for håndtering af det der skal ske på skærmen via TFT driveren. Det kan f.eks. være hvordan den skal flytte et grafisk object.

## Sekvensdiagrammer

## GameEngine



Figur 7- sd GameEngine – Dette diagram giver en oversigt over de mest basale kommandoer i GameEngine, spawn, move, shoot og tick. – spawn: er den kommando der bruges til at oprette et nyt GameSubject. Det første skridt er at bruge metoden spawn til at fortælle GameEngine, at der ønskes et ny GameSubject af typen hero, herefter vil GameEngine bruge en Factory til at oprette det ønsket GameSubject. Når et GameSubject bliver oprettet vil der også blive oprette et tilhørende GFXObject, som er et grafisk element på skærmen, derfor bliver der krævet en tilskrivning til skærm kontrollen. Hvis der ikke er opstået et hukommelse fejl vil factory’en returner en pointer til den nye GameSubject, som GameEngine vil gemme i et array hvor dens index vil blive brugt som ID, der også vil blive sendt tilbage til aktoren. – Den næste kommando er move som bruges til at flytte et GameSubject, det er muligt at vælge at flytte GameSubjectet udfra origo, eller den relative position for GameSubjectet selv. Denne metode vil kun ændre et par variabler i GameSubject, og vil ikke få GameSubjectet til at flytte sig. Dette sker først når et tick er kørt. – Shoot kommandoen vil oprette et skyd der kommer fra det GameSubject der henvist til. Denne metode fungere på samme måde som spawn, udover GameSubjectet bliver lagt i et andet array kun for skyd. Disse shoot GameSubject kan ikke styres af Aktoren, men bliver automatisk styret af GameEngine i hvert tick. – Den sidste metode der bliver kaldt fra Aktoren er tick. Det første der sker i et tick er, at den kontrollere om der er et GameSubject der er ramt af fjendeligt skud. Hvis ja vil GameSubjectet miste et liv, og skuddet vil blive slettet. Det næste skridt er så at opdatere alle GameSubject’ers positioner; her vil den gennemløbe alle GameSubjects og opdatere deres grafiske element så det passer med GameSubjectet. Til sidst vil GameEngine opdatere skyd positionerne.

### GFX – Jonathan

# All test results

F.eks. debug ben til touch.

Analogdiscovery dump af analog pins.

## TFT TEST –Jonathan?

## TOUCH TEST – Jonathan?

1. <http://minecraft.gamepedia.com/Tick> (Det eneste sted som havde en fin beskrivelse af hvad et tick var.) [↑](#footnote-ref-1)
2. Universal Digital Block : <http://www.cypress.com/video-library/PSoC/intro-udb-components-psoc-5lp/386161> [↑](#footnote-ref-2)
3. Terminal Count: <http://www.cypress.com/documentation/component-datasheets/timer> [↑](#footnote-ref-3)
4. <http://www.cypress.com/documentation/development-kitsboards/cy8ckit-042-psoc-4-pioneer-kit> [↑](#footnote-ref-4)