Software arkitektur



Figur 1 - klassediagram for battleships projektet

På figur 1 kan ses et klassediagram over projektet, ikke alt fra koden er inkluderet i klassediagrammet, for at få et bedre overblik.

bitmaps er en header fil der indeholder bitmaps i 16bit farve format i arrays der bliver gemt i program hukommelsen, dette bliver brugt til at tegne figurer på skærmen.

bitmaps bliver brugt i TFT Driver, som er skærm driveren, da det er TFT driver der tegner på skærmen. Den består af DisplayInit som sætter alt op til at bruge skærmen som pixelformat m.m. derudover er der drawBitmap der tegner et bitmap fra bitmaps filen på skærmen. FillRectangle tegner en firkant i en solid farve på skærmen. drawGameboardYLines og drawGameboardXLines bruges i drawGameboard i gameboard klassen til at tegne gitteret på skærmen.

Touch Driver er der vi får vores input, den består af touchInit der sætter alt op til brug af touch, touchMeassure laver en måling og gemmer den i et x og y koordinat, den gør brug af touchRead og touchWrite til at kommunikere med touch hardware modulet. touchReady fortæller om der er nyt data fra touch modulet og bruges til at fortælle spillet om der er trykket på knappen. ButtonPressed bruge x og y koordinatet til at finde ud af hvilket knap der er trykket på og retunere en karakter der repræsentere en af knapperne.

Helper functions er lavet som et pseudo objekt, den har funktioner makeCord, addCord, getXCord, getYCord, compareXCord og compareYCord der bruges til at arbejde koordinater, da vores spille areal ikke ville være større en 16x16 kunne man bruge en uint8\_t til at holde koordinaterne de fire mindst betydende bit til x koordinatet, og de fire mest betydende bit til y koordinatet. Så makeCord tager et x og et y koordinat og repræsentere det i en enkelt uint8\_t. addCord er en funktion man kan bruge til at tælle x eller y koordinatet en op eller ned, den bliver blandt andet brugt til at flytte skibene der er repræsenteret med et sæt af koordinater. getX(Y)Cord returnere den respektive x eller y fra et koordinat og compareX(Y)Cord retunere om den ene koordinat er enten større end, mindre end eller lig med hindanden.

Gameboard er det eneste rigtige objekt i koden, den består af to array af arrays der repræsentere spillerens areal og modstanderens areal, derud over er der en variable til at holde styr på hvor mange skiber hver spiller har ramt, missileHits for spilleren og cpuMissiles for den virtuelle modstanderen. Dertil kommer funktionerne, placeShip tager en pointer til et array med koordinater til båden, og en længde af båden, og en boolean til at fortælle om det er spilleren eller modstanderen der sætter båden. drawGameboard bruger TFT driveren til at tegne gameboardet som det er, den tager også en pointer med til et array med koordinaterne til båden og længden sådan at man kan tegne hvor båden er uden at have placeret den endnu. startGame skal kaldes når spilleren har sat alle sine både, den skifter en boolean sådan at programmet ved at knapperne skal gøre noget andet, og så sætter den bådene for den virtuelle modstander. Hit er funktionen der skyder missilerne afsted og tjekker om den ramte noget hvis den gjorde tæller den missileHits eller cpuMissiles op alt efter hvem skød dette er indikeret med en boolean parameter. Den tjekker også hvorvidt der er ramt 9 både og hvis der er kalder den, den sidste funktion won som stopper spillet og sætter skærmen til grøn eller rød alt efter om spilleren eller modstanderen vandt og skriver ”WIN” eller ”LOSS” på skærmen.

Det meste logik bliver håndteret af gameboard, dog er det i main man opretter objektet og også i main at man registrere hvilken knap der bliver trykket på og så bare kalder functioner fra gameboard alt efter knappen

Tekniske overvejelser

Der har været mange overvejelser i projektet der blev tidligt i projektet valgt at der ville gøres brug af c++ og klasser til vores spil logik, der var oprindeligt planner om at der skulle være 3 klasser, game klassen der ville bestå af 2 gameboard klasser, og en ship klasse der ville bruges i gameboard.

Game klassen ville have to gameboard klasser et til at repræsentere spiller et og et til spiller to. Dette blev dog ændret til at gameboard ville indeholde information om både spilleren og modstanderens gameboard.

Ship klassen blev også fuldt implementeret men der opstod problemer med klassen så den ikke kunne kompileres efter en masse forsøg på at løse dette blev der valgt at lave en alternativ til ship klassen der bestod af en håndfuld funktioner, initShip der initieret et array bestående af koordinater til at repræsentere hvor skibet var, og funktioner til at rykke på disse koordinater sådan at man kunne bruge userinterface til at rykke rundt på skibet, med funktionerne up(), down(), left(), right() og rotate(). Så havde gameboard en funktion der ville tage dette array og placere det på gameboardet.

Der var oprindeligt planer om at dette skulle være et to spiller spil, der kunne spilles mellem to atmega2560, dette blev dog droppet grundet både hardware og software problemer som der vil beskrives nærmere i det næste afsnit. Som et alternativ blev der valgt at implementere en computer spiller som ville prøve at sætte skibene tilfældigt og derefter gætte tilfældigt når den skød. Til dette ville der gøres brug af en indbygget funktion rand() i stdlib. Denne funktion viste sig dog at være lidt mere forudsigelig en først forventet. Der var både problemer med at den ikke var tilfældig, altså resultatet blev det samme hver gang programmet kørte, og så at compileren ville optimere variablen det tilfældige tal var gemt i væk. Efter at optimering blev slået fra blev resultat umiddelbart nyt, dog anden gang det blev kørt var det tilbage til at være forudsigeligt. Der blev valgt at fokusere på andre problemer, og optimering blev sat til igen for at få spillet til at reagere bedre.

Til projekt var der gjort forsøg på at implementere radio kommunikation, og selvom driveren til dette blev skrevet, er det ikke blevet testet med modulet, da modulets pins ikke passede med nogle kabler der var til rådighed. Som et alternativ, prøvede der at implementeres SPI, da radiodriveren gjorde brug af det. SPI virkede i test, men var inkonsekvent og efter en masse test på SPI, blev der valgt at droppe kommunikation mellem mikrocontroller, da dette også betød at der kunne undgås at skulle implementere et realtime operating system. Da der kun sker noget i spillet når der bliver trykket på en knap kan der nøjes med at opdatere nå der bliver trykket på en knap. Dog hvis der var blevet gjort brug af SPI eller radio, ville det have blokeret helle programmet hver gang der skulle ventes på et svar. Så for at håndtere dette ville det have været en fordel at bruge threads til at sørge for at spillet altid ville reagere på bruger input. Dette blev alt sammen undgået ved at droppe kommunikation.

Der er ikke gjort de store forsøg på at optimere spillet, f.eks. hver gang vi opdatere skærmen bliver alt opdateret ikke kun det der har ændret sig, det havde været en fordel hvis der kun blev opdateret på ændringer der blev lavet siden sidst, og at statiske UI-elementer altid var på skærmen. Det har været en højere prioritet i dette projekt at have et spil der virker, og der er derfor lagt mere arbejde i spil logikken end optimeringen.

Alternative løsninger

Der var originalt planer om at projektet skulle inkludere 2 Arduino mega2560 der kommunikerede trådløst med hinanden, der blev lånt 2 radio moduler der var planer om skulle bruges til at komunikere i mellem dem, dette havde selvfølgelig den fordel at der kunne spilles mellem to spiller over en afstand.

Dog blev radio droppet som en mulighed da modulerne havde mindre afstand mellem pins betød det at de kabler der var tilgængelige ikke passede på dem, derfor blev der set på andre alternativer, som f.eks. SPI, UART og I2C, alle disse løsninger havde dog en ting tilfældes, det ville kræve at vi implementeret freeRTOS, eller et andet real time operating system, til at håndtere tråde, mikrocontrolleren ellers ville blokere så længe de ventede på at få svar fra modstanderen.