AARHUS UNIVERSITY
AARHUS UNIVERSITY SCHOOL OF ENGINEERING

Tic Tac Toeminator

Semesterprojektgruppe: 8

Arkitektur og design

Studienummer:	*Deltager:*
1# 201609213	Jakob Rune Skov
2# 201608892	Jesper Mundbjerg Madsen
3# 201607598	Mathias Kirkeby
4# 201605348	Morten Dahl Nielsen
5# 201605961	Morten Rahr Nielsen
6# 201608798	Simon Fabricius Nielsen
7# 201609922	Tobias Saaby Steffensen
8# 201600313	Rune Bjørn Lassen

Vejleder:
Peter Høgh Mikkelsen

22.12.17

Indholdsfortegnelse

Figurer

Tabeller

Ordforklaring

Forkortelse:	Forklaring:
Ack	Acknowledgement
APSoC	A = Arm, PSoC-styring til arm
Arm	Robotarm
ASCII	American Standard Code for Information Interchange
BDD	Block Definition Diagram
CD	Class Diagram
CPU	Central Processing Unit
DB	Database
DHCP	Dynamic Host Configuration Protocol
EFYS	Elektrofysik
EM	Elektromagnet
FM	Finite Mode, spillemode til pladen fuld
GPIO	GeneralPurpose Input/Output
GUI	Graphical User Interface
GUIrpi	RPi, hvorpå GUIprogram ligger
HDMI	High-Definition Multimedia Interface
HW	Hardware
IBD	Internal Block Diagram
IF	InterFace
IM	Infinity Mode, spillemode med tre brikker til hver spiller
IRQ	Interrupt Request
IP	Internet Protokol
KB	Kryds og bolle

Forkortelse:	Forklaring:
Master	En RPi som styrer systemet.
MAC	Media Access Control
MoSCoW	Must have, Should have, Could have, and Won't have
MSB	Most Significant Bit
PLA	Polylactic Acid
PNP	Positive-Negative-Positive
PPSoC	P = Plade, PSoC-styring til spilleplade
PSoC 5LP	Programmable System-on-Chip
PWM	Pulse-Width Modulation
QML	Qt Meta Language eller Qt Modeling Language
RPi	Raspberry Pi Zero W, single-board computer device
SD	Sequence Diagram
SPI	Serial Peripheral Interface
SQL	Structured Query Language
STM	State Machine Diagram
SW	Software
TTT	Tic Tac Toeminator
UART	Universal Asynchronous Receiver/Transmitter
UC	Use Case Diagram
Wi-Fi	Trådløst netværk

Arkitektur

I dette dokument følger en beskrivelse af projektets overordnede arkitektur. Først gives en introduktion til systemets arkitektur med en beskrivelse af systemets overordnede blokke og deres funktioner samt allokering af software på de forskellige blokke.

Herpå følger en beskrivelse af hardware-arkitekturen, hvor systemet neddeles i hardwareblokke ved brug af Blok Definitions diagrammer (BDD). Disse blokkes interne forbindelser vises på Interne Blok diagrammer (IBD), der følges op med en detaljeret signalbeskrivelse af forbindelsernes navne, typer og niveauer.

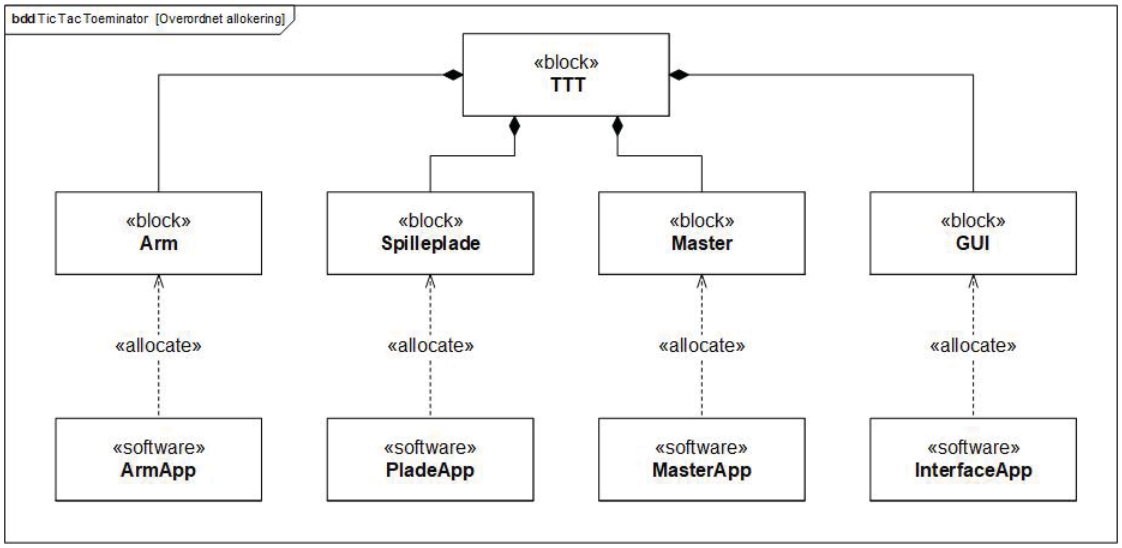
Herefter gennemgås software-arkitekturen, hvor systemets software beskrives ved hjælp af applikationsmodeller, som indbefatter klassediagrammer (CD) og sekvensdiagrammer (SD). Diagrammerne bruges til at undersøge hvilke klasser og funktioner der skal implementeres.

Senere i dokumentet følger en udspecificering af systemets design, som ligeledes opdeles i hardware og software. Hardware designet indeholder en beskrivelse af alle blokkene, som er identificeret og specificeret i arkitekturen. For hver blok opstilles der kredsløbsdiagrammer, hvor teorien, formler og beregninger medtages.

Software designet indbefatter opdaterede klassediagrammer med attributter og metoder. For de mest betydningsfulde metoder er der lavet funktionsbeskrivelser med navn, returtype, parameter og funktionalitet. Som supplement tilføjes der tilstandsdiagrammer (STM), hvor det er hensigtsmæssigt for forståelsen.

1 System arkitektur

Systemet kan overordnet inddeles i fire delsystemer, som er vist på Figur 1. Et delssytem defineres som et system med egen "CPU". Det vil sige, at der skal udvikles en software applikation til hver af de fire blokke. På figuren vises også allokeringen af den software, som skal anvendes på de enkelte delsystemer. Stereotypen <<allocate>> bruges på associationer fra software- til hardware blokke, for at vise hvilken software applikation, der hører til hvilken hardware blok. Figur 1 giver overblik over hvilke delsystemer og hvilke software applikationer, som skal udvikles.



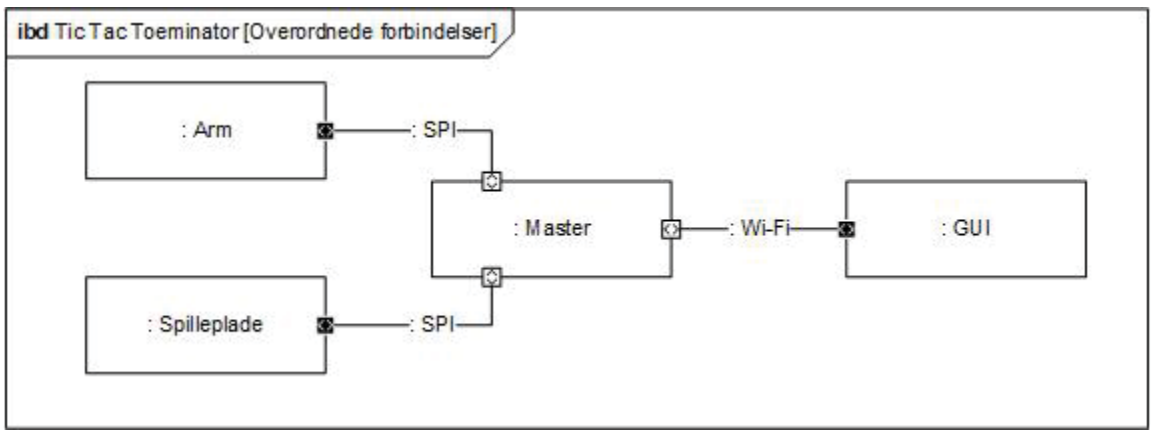
Figur 1: Overordnet Blok Definitions diagram med softwareallokering

I Tabel 1 følger en kort beskrivelse af hver bloks formål og funktion.

Arm	Armen er den del af systemet, der varetager den fysiske interaktion med spillets brikker. Armens funktion er at bevæge spillets brikker mellem to givne koordinater på Spillepladen. Armen består - foruden en række mekaniske komponenter - af en elektromagnet, 3 servomotorer og en PSoC. Servomotorerne varetager bevægelsen af Armen. Elektromagneten hhv. opsamler og slipper de metalliske spillebrikker på de ønskede steder. PSoC'en varetager styring af Armens servomotorer og elektromagnet samt kommunikation med Masteren.
Spilleplade	Spillepladen består af en PSoC og 9 induktive sensorer. Spillepladen er udformet med 9 felter, som hver har én induktiv sensor tilsluttet. PSoC'en står for kommunikation med Masteren og overvåger, gennem de induktive sensorer, hvor på pladen brugeren placerer sine brikker. Masteren kan få PSoC'en til at aflæse sensorerne og sende en status tilbage.
Master	Masteren består af en RPi, der varetager kommunikation med systemets 3 øvrige blokke samt afvikling af den del af systemets software der foretager spil-beregningerne. På baggrund af input fra Spillepladen og brugergrænsefladen, analyserer og beregner Masteren hvordan spillet skrider frem. Masteren indhenter informationer fra Spillepladen og afgører om spillet er afsluttet (Win/Loose/Draw), hvis ikke, beregner Masteren et nyt træk, som den sender til Armen.
GUI	Brugergrænsefladen består af en Rasberry Pi med en tilknyttet touchskærm. Brugeren starter spillet, indstiller sværhedsgrad m.v. gennem interaktion med touchskærmen. Rasberry Pi'en sender relevante informationer til Masteren, så spil-beregningerne kan ske i henhold til brugerens valg.

Tabel 1: Beskrivelse af det overordnede Blok Definitions diagram

De overordnede forbindelser og det logiske flow mellem delssystemerne er beskrevet på IBD'et, Figur 2. Her vises de to protokoller (Seriel og Wi-fi) som delsystemerne benytter sig af. Protokollerne er yderligere beskrevet i afsnit 4 Software Protokoller.



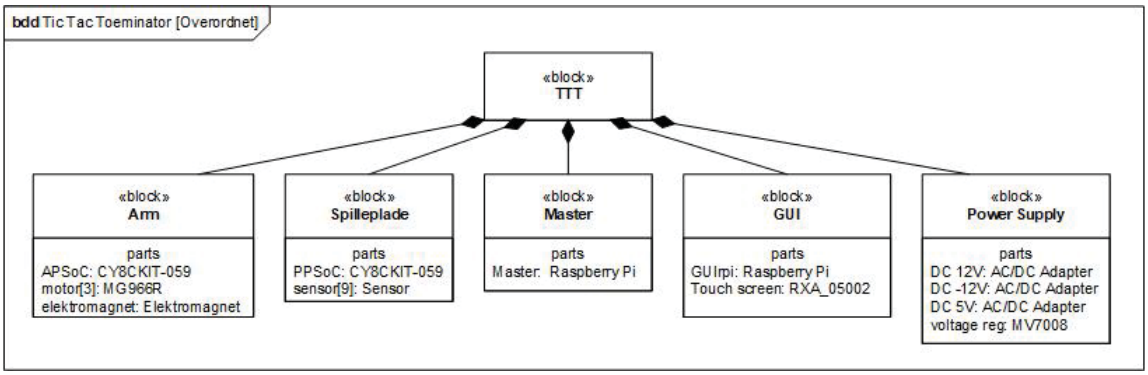
Figur 2: Overordnet Internt Blok diagram

2 Hardware arkitektur

I de følgende afsnit vil hardware arkitekturen for systemet blive beskrevet. Først indledes afsnittet med et overordnet Blok Definitions diagram og derefter delblokke for systemets mindre dele (Figur 3 til 6). Hertil er der tilhørende tabeller med blokbeskrivelse af systemets enkelte dele (Tabel 2 til 5). De interne signaler mellem systemets blokke vises på Interne Blok diagrammer, først for det overordnede system og derefter for de mindre blokke (Figur 7 til 10). Alle signalerne mellem systemets blokke beskrives efterfølgende i tabeller med signalbeskrivelser (Tabel 6 til 9). Systemets delblokke er følgende: Arm, Spilleplade og GUI.

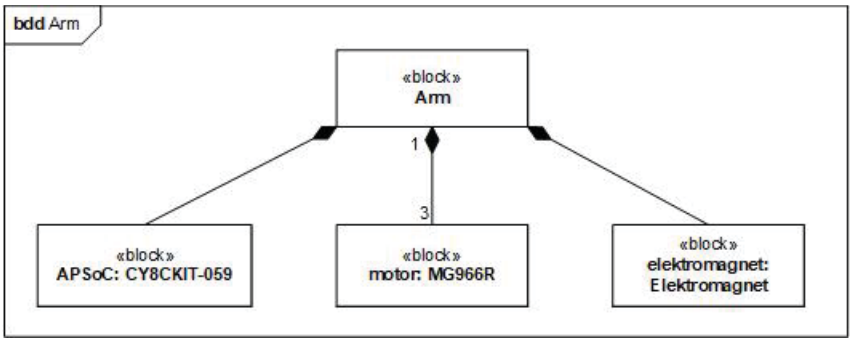
2.1 Blok Definitions diagrammer

Ud fra en videre analyse af systemet vist på Figur 1, er der fundet frem til at de overordnede blokke indeholder mindre delblokke. På Figur 3 vises det overordnede BDD, hvortil der er tilføjet blokken Power Supply, som står for forsyningen af hele systemet.

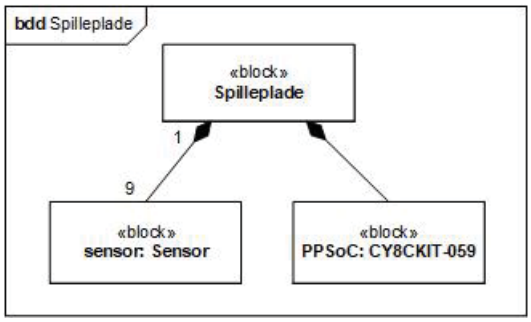


Figur 3: Overordnet Blok Definitions diagram for TTT

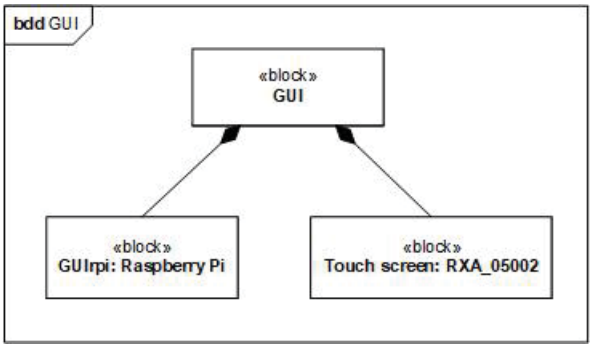
Herunder er der zoomet ind i blokkene Arm, Spilleplade og GUI, hvor delblokkene vises, som hver beskriver en mindre del af systemet.



Figur 4: Blok Definitions diagram for Armen



Figur 5: Blok Definitions diagram for Spillepladen



Figur 6: Blok Definitions diagram for GUI

2.2 Blokbeskrivelse

Blokbeskrivelsen beskriver blokkene og delblokkene i Blok Defintions diagrammerne vist på Figur 3 til 6. Hver blok er angivet med blok-navn, en beskrivelse af blokkens funktionalitet, samt in og out signaler.

Blok-navn	Funktionsbeskrivelse	Signaler	Kommentar
Master	Modtager og analyserer data fra de andre enheder i systemet. Udregner træk på baggrund af data og styrer Arm.	wi-fi: Wi-Fi	Trådløs kommunikation mellem Master og GUI
		armCtrl: SPI	SPI kommunikation mellem Master og Arm
		pladeCtrl: SPI	SPI kommunikation mellem Master og Spilleplade
		MasterS: 5V	Spændingsforsyning til Master
GUI	Modtager input fra bruger og viser GUI til bruger.	touch: Touch	Input fra bruger
		wi-fi: Wi-Fi	Trådløs kommunikation mellem Master og GUI
		display: Display	Display af GUI
		GUIrpiS: 5V DC	Spændingsforsyning til GUI
		ScreenS: 5V DC	Spændingsforsyning til Touch screen
Power Supply	Forsyner alle andre enheder med spænding.	MasterS: 5V DC	Spændingsforsyning til Master
		GUIrpiS: 5V DC	Spændingsforsyning til RPi i GUI
		PPSoCS: 5V DC	Spændingsforsyning til pladePSoC
		APSoCS: 5V DC	Spændingsforsyning til armPSoC
		armS: 5V DC	Spændingsforsyning til elektromagnet og servomotorer
		230V: 230V AC	Spænding fra el-nettet
		ScreenS: 5V DC	Spændingsforsyning til Touch screen
Spilleplade	Detekterer brikker og sender information til Master	pladeCtrl: SPI	SPI kommunikation mellem Master og Spilleplade
		PPSoCS: 5V DC	Spændingsforsyning til pladePSoC
		Spilleplade+: 12V DC	Positiv spændingsforsyning til komponenter i sensorkreds.
		Spilleplade-: -12C DC	Negativ spændingsforsyning til komponenter i sensorkreds.
Arm	Flytter brikker.	armCtrl: SPI	SPI kommunikation mellem Master og Arm
		APSoCS: 5V DC	Spændingsforsyning til APSoC
		armS: 5V DC	Spændingsforsyning til elektromagnet og servomotorer

Tabel 2: Blokbeskrivelse af blokkene på Blok Definitions diagrammet for TTT

Blok-navn	Funktionsbeskrivelse	signaler	Kommentar
motor[0..2]: MG966R	Drejer akser på Armen	armS: 5V DC	Spændingsforsyning til servo-motorer
		pwm: PWM	PWM-signal til at bestemme vinkel til motor
APSoC: PSoC	Kommunikerer med Master for at styre armens motorer samt elektromagneten	pwm: PWM	PWM-signal til at bestemme vinkel til motor
		∼armCtrl: SPI	SPI kommunikation mellem Master og Arm
		EM: LVTTL	Signal til aktivering af elektromagnet
		APSoCS: 3.3V	Spændingsforsyning til armPSoC
Elektromagnet	Opsamler og slipper spillebrikker	armS: 5V DC	Spændingsforsyning til elektromagnet
		EM: LVTTL	Signal til aktivering af elektromagnet

Tabel 3: Blokbeskrivelse af blokkene på Blok Definitions diagrammet for Armen

Blok-navn	Funktionsbeskrivelse	Signaler	Kommentar
PPSoC: PSoC	Monitorerer sensorer og sender data til Master	∼pladeCtrl: SPI	SPI kommunikation mellem Master og Arm
		PPSoCS: 5V	Spændingsforsyning til plade-PSoC
Sensor[0..8]: Sensor	Opfanger hvorvidt der er en brik på pladsen vha. induktion.	: Analog	Analogt signal fra sensorkredsen der samples på PSoC'en
		Spillleplade+: 12V	Positiv spændingsforsyning til komponenter i sensorkredsen.
		Spilleplade-: -12C	Negativ spændingsforsyning til komponenter i sensorkredsen.

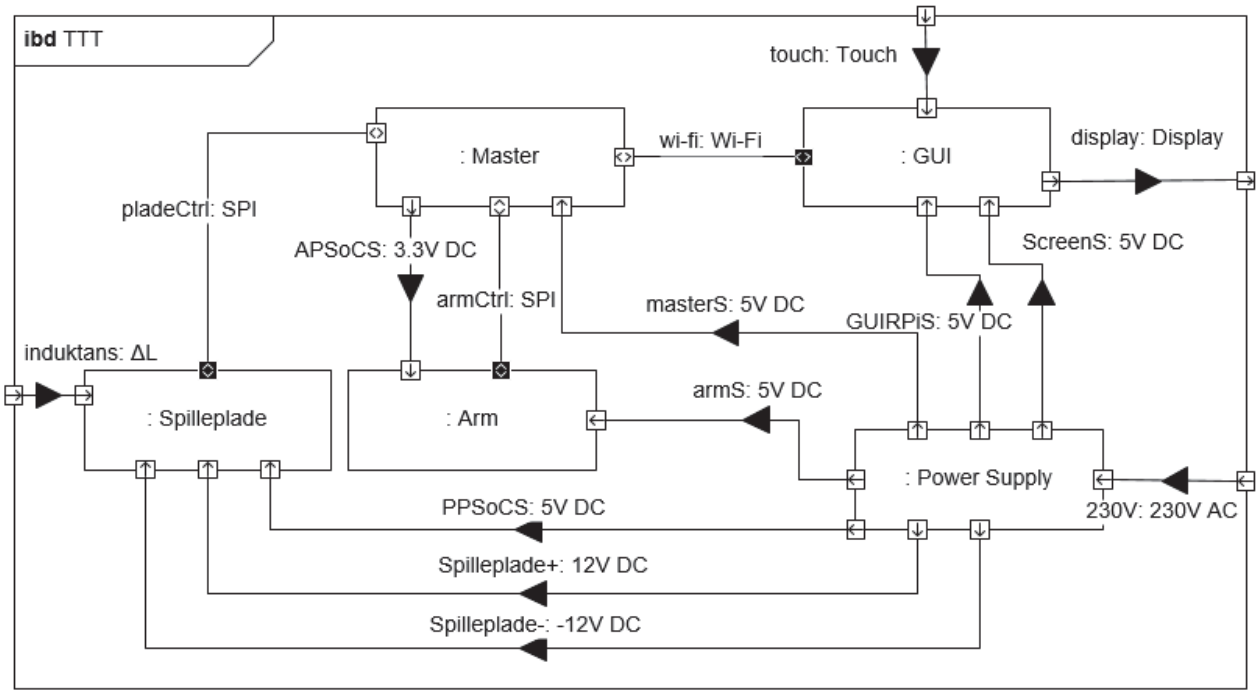
Tabel 4: Blokbeskrivelse af blokkene på Blok Definitions diagrammet for Spillepladen

Blok-navn	Funktionsbeskrivelse	signaler	Kommentar
Touch screen	Fungerer som interface med brugeren.	touch: Touch	Input fra bruger
		display: Display	Display af GUI
		ScreenS: 5V DC	Spændingsforsyning til Touch screen
		hdmi: HDMI	Til kommunikation mellem GUIrpi og Touch screen
GUIrpi: Raspberry Pi	Modtager input fra bruger og viser interfacet til brugeren vha. Touch screen.	wi-fi: Wi-Fi	Trådløs kommunikation mellem Master og GUI
		GUIrpiS: 5V	Spændingsforsyning til GUI
		hdmi: HDMI	Til kommunikation mellem GUIrpi og Touch screen

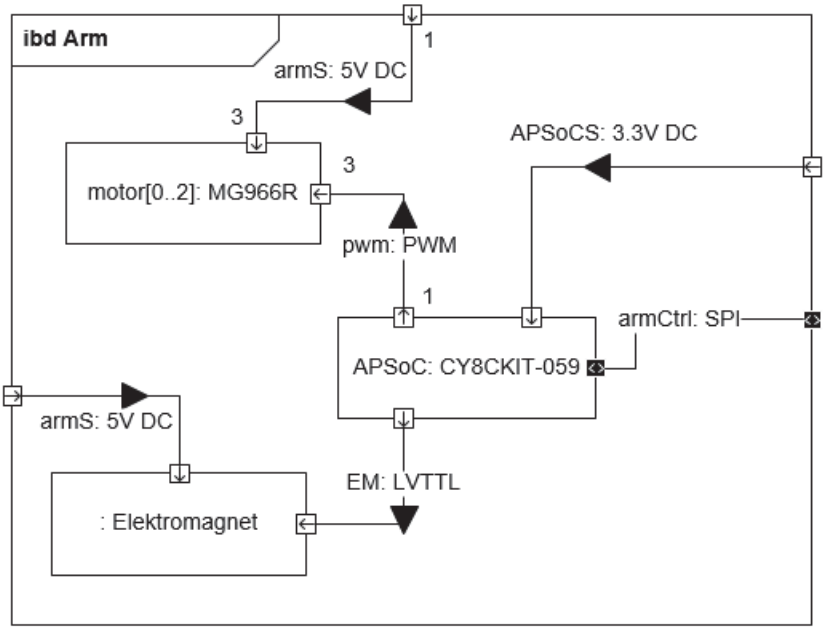
Tabel 5: Blokbeskrivelse af blokkene på Blok Definitions diagrammet for GUI

2.3 Interne Blok Diagrammer

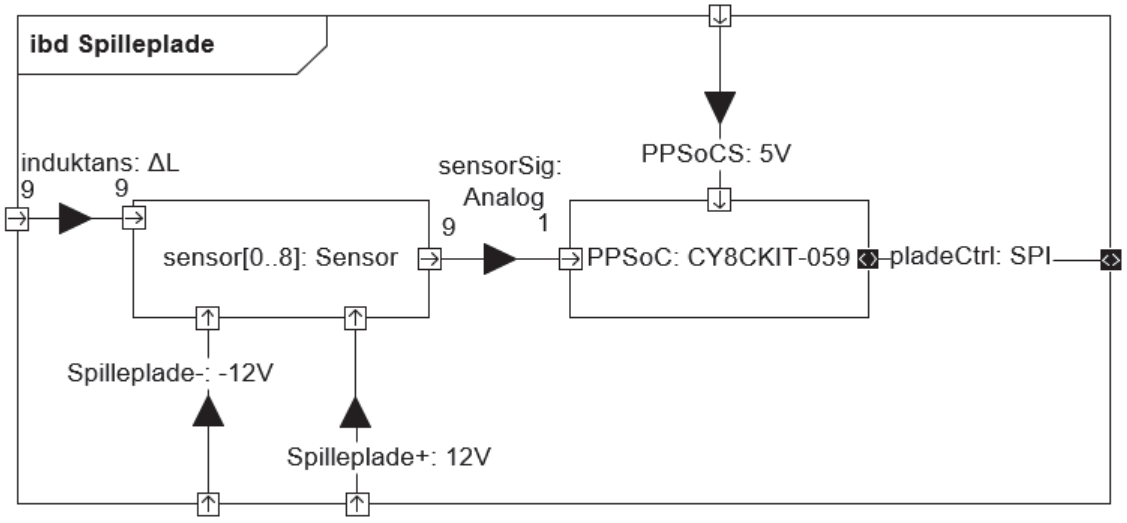
De Interne Blok Diagrammer viser TTT systemets interne signaler.



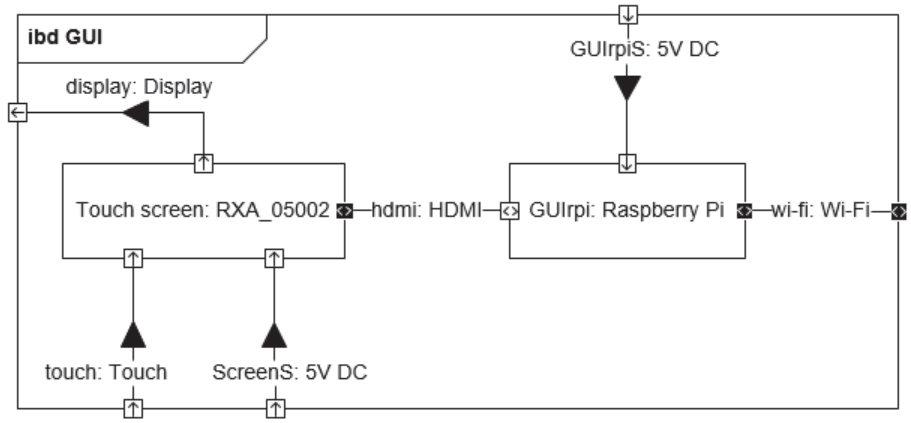
Figur 7: Internt Blok diagram for TTT med signaler



Figur 8: Internt Blok diagram for Armens 3 servomotorer og elektromagneten



Figur 9: Internt Blok diagram for Spillepladens 9 sensorer



Figur 10: Internt Blok diagram for GUI

2.4 Signalbeskrivelse

Signalbeskrivelsen beskriver diverse signalers funktioner, størrelser og områder i de Interne Blok diagrammer. I tabellen angives fysiske pins på en PSoC med P. Placeringen af pins kan ses i databladet for CY8CKIT-059, der findes i bilag. Pins på en RPi angives med j og deres placering kan ses i databladet for rpizw, som også findes i bilag.

Signal-navn	Funktion	Område	Fysiske porte	Kommentar
wi-fi: Wi-Fi	Til trådløs kommunikation mellem Master og GUI			
armCtrl: SPI	SPI kommunikation mellem Master og Arm	0V-3.3V	RPi: ss: J8.24 mosi: J8.19 miso: J8.21 sck: J8.23 Int: J8.36 PSoC: ss: P1.7 mosi: P1.6 miso: P1.5 sck: P1.4 Int: P1.2	SPI-protokollen er beskrevet i afsnit 4.1
pladeCtrl: SPI	SPI kommunikation mellem Master og Plade	RPi: 0V-3.3V PSoC: 0V-5V	RPi: ss: J8.26 mosi: J8.19 miso: J8.21 sck: J8.23 Int: J8.32 PSoC: ss: P1.7 mosi: P1.6 miso: P1.5 sck: P1.4 Int: P1.2	SPI-protokollen er beskrevet i afsnit 4.1. Der er anvendt en logic-level-converter mellem PSoC og rpi. Opsætningen af denne er beskrevet i afsnit 6.3.1
touch: Touch	Input fra bruger			
display: Display	Display af GUI			
MasterS: 5V DC	Spændingsforsyning til Master	4.75V-5.25V	RPi: J8.2	
GUIrpiS: 5V DC	Spændingsforsyning til RPi i GUI	4.75V-5.25V	RPi: J8.2	
PPSoCS: 5V DC	Spændingsforsyning til pladePSoC	4.75V-5.25V	PSoC: VDDIO	
ScreenS: 5V DC	Spændingsforsyning til Touch screen	4.75V-5.25V		
APSoCS: 3.3V DC til arm PSoC	Spændingsforsyning til	3.1V-3.5V	PSoC: VDDIO	
armS: 5V DC	Spændingsforsyning til elektromagnet og servomotorer	4.8V-5.2V	servo: Vcc	
230V: 230V AC	Spænding fra el-nettet	230V AC		

Tabel 6: Signalbeskrivelse af det Interne Blok diagram for TTT

De følgende signalbeskrivelser er for de resterende delblokke og indeholder kun en beskrivelse af de signaler, der ikke allerede er beskrevet under signalbeskrivelsen for TTT.

Signal-navn	Funktion	Område	Fysiske porte	kommentar
EM: LVTTL	Signal til aktivering af elektromagnet	0V-3.3V	P1.2	
pwm: PWM	PWM-signal til at bestemme vinkel til motor	0V-3.3V	P2.2[0], P2.3[1], P2.4[2], Motor[0]PWM, Motor[1]PWM, Motor[2]PWM	

Tabel 7: Signalbeskrivelse af det Interne Blok diagram for Armen

Signal-navn	Funktion	Område	Fysiske porte	kommentar
sensorSig : Analog	Analogt signal, som viser, om der er en spillebrik til stede eller ej		P3.0	

Tabel 8: Signalbeskrivelse af det Interne Blok diagram for Spilleplade

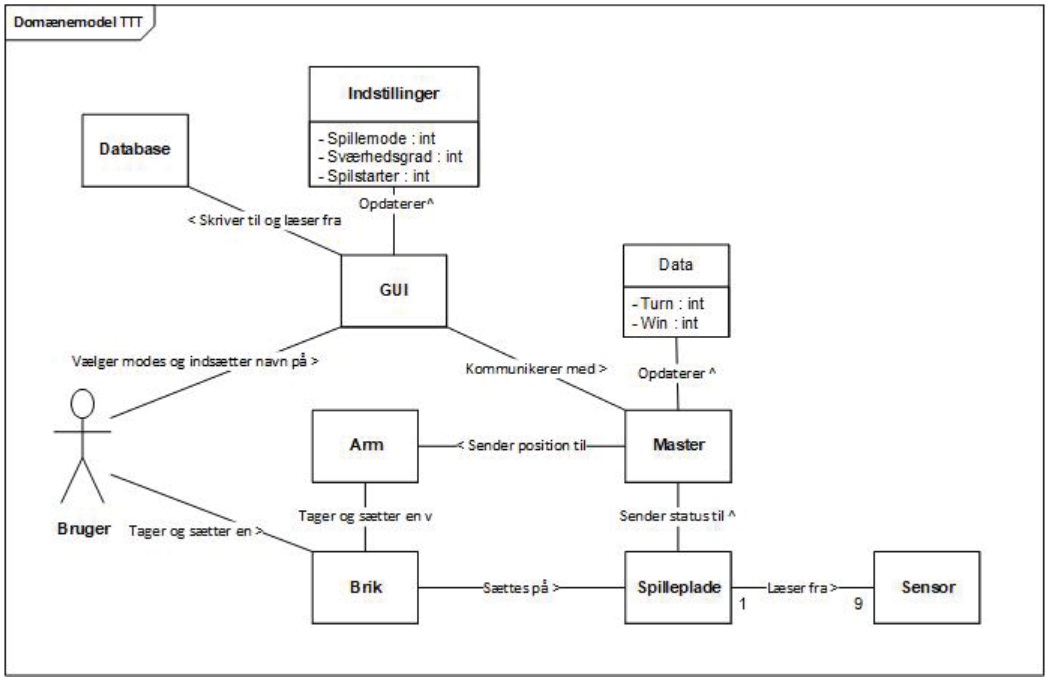
Signal-navn	Funktion	Område	Fysiske porte	kommentar
hdmi: HDMI	Til kommunikation mellem GUIrpi og Touch screen			

Tabel 9: Signalbeskrivelse af det Interne Blok diagram for GUI

3 Software arkitektur

I de følgende afsnit vil software arkitekturen for systemet blive beskrevet. Der indledes med en domænemodel, som følges af klassediagrammer og sekvensdiagrammer lavet ud fra vores Use Cases, som kan findes i dokumentet Kravspecifikation afsnit 3.1.

På Figur 11 ses domænemodellen, som har til formål at skabe overblik over systemet.



Figur 11: Domænemodel for TTT

GUI'en opdaterer indstillingerne til at passe med brugerens ønsker. Vi bruger 3 int's for de forskellige valgmuligheder, spillemode, sværhedsgrad og spilstarter. Dette er vigtigt, da GUI'en skal sende disse indstillinger til master, men samtidig huske indstillingerne når den skal opdatere databasen. Masteren opdatere træk, som har int turn, som skal sendes til GUI. Dette er vigtigt, da dette er den score, som brugeren har opnået.

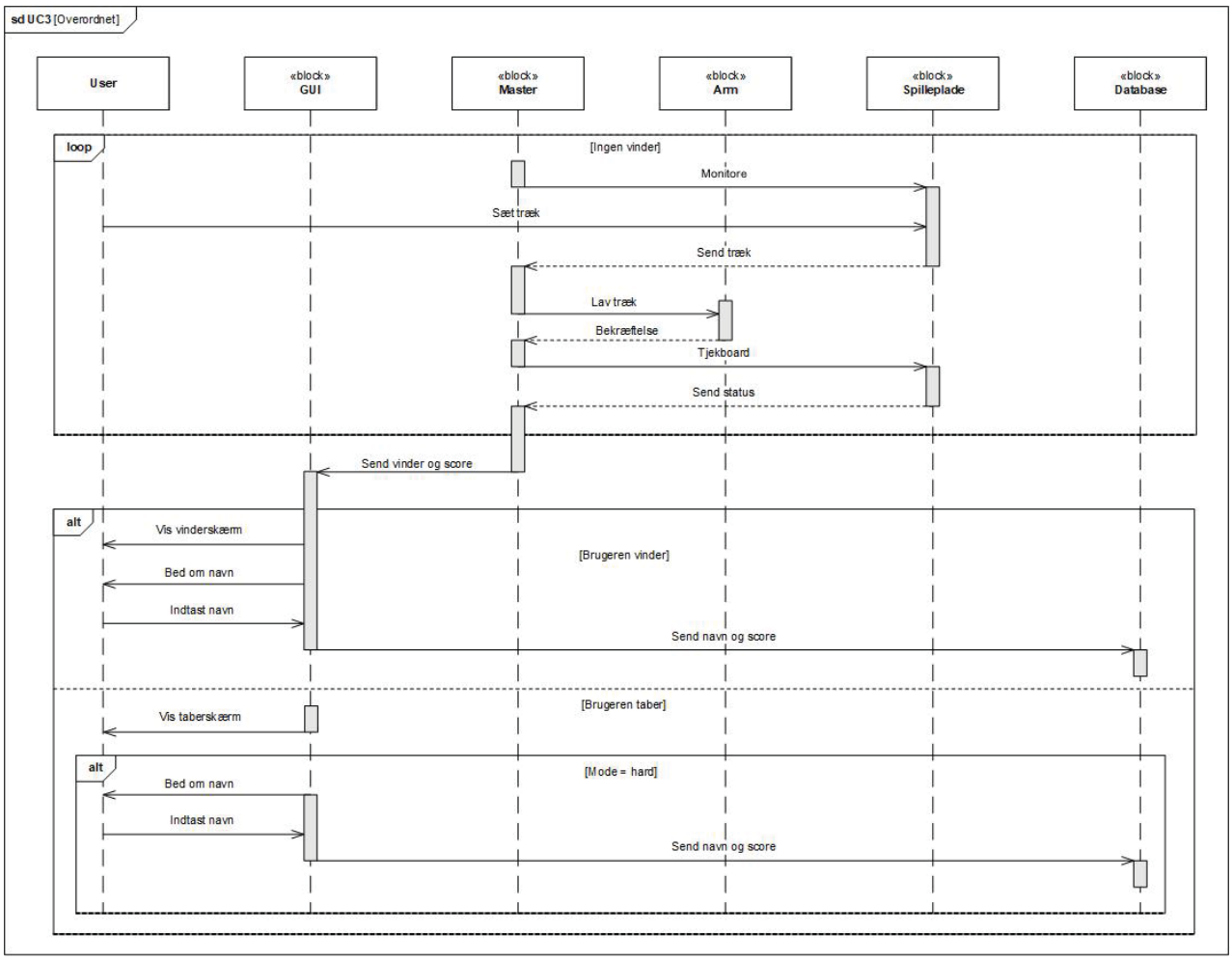
3.1 Softwareallokering

Som der kan ses på Figur 1 i kapitlet 1 System Arkitektur, skal der allokeres software til systemets hardwarekomponenter:

- ArmApp: Armen skal have software, der skal styre måden hvorpå Armen skal bevæge sig, og hvilke koordinater Armen skal bevæge sig til. Det er også dette stykke software, der skal styre elektromagneten.
- PladeApp: Dette er applikationen, som er allokeret på PSoC'en tilhørende Spillepladen, og derfor den som skal detektere brikkerne på Spillepladen.
- MasterApp: Er applikationen der er allokeret på Masteren. Dette er "hjernen" i systemet, og den der skal holde styr på de andre applikationer i systemet. Det er også dette stykke software hvorpå minimax-algoritmen kommer til at køre.

- InterfaceApp: Dette stykke software bliver kørt på user interface. Det er udformet som et grafisk stykke software, som skal skabe kommunikation mellem brugeren og Master.

Hvis man ser på Figur 12 kan man se et sekvensdiagram for kommunikationen mellem blokkene for Use Case 3. Grunden til, at denne Use Case er valgt, er, at den viser det meste af kommunikationen mellem blokene. På sekvensdiagrammet er databasen vist som en selvstændig blok. Herved kan man foranlediges til at tro, at databasen har sin egen RPi, hvilket ikke er tilfældet. Databasen ligger som en del af GUI'en, men beskrives af forståelsesmæssige grunde som en selvstændig blok på sekvensdiagrammet.



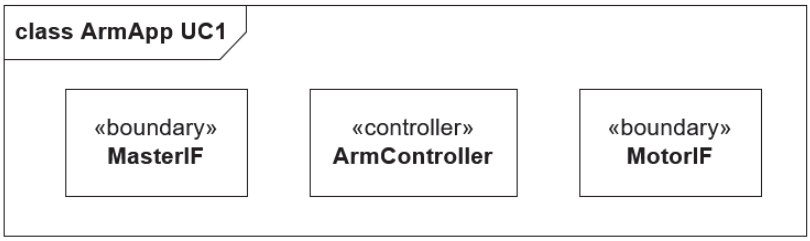
Figur 12: Kommunikation mellem blokke for Use Case 3

3.2 Applikationsmodel ArmApp

I det følgende afsnit beskrives den overordnede software arkitektur for ArmApp ved brug af klassediagrammer og sekvensdiagrammer. Koden til ArmApp er skrevet i sproget *C*, så softwareblokkene i diagrammerne er moduler, ikke klasser.
Når der i et diagram står *ack* skal det forstås som en returværdi, der validerer, om funktionen har fået gyldige parametre eller om noget gik galt under eksekvering. Hvis alt gik godt returneres 1, hvis ikke returneres -1.

Use Case 1

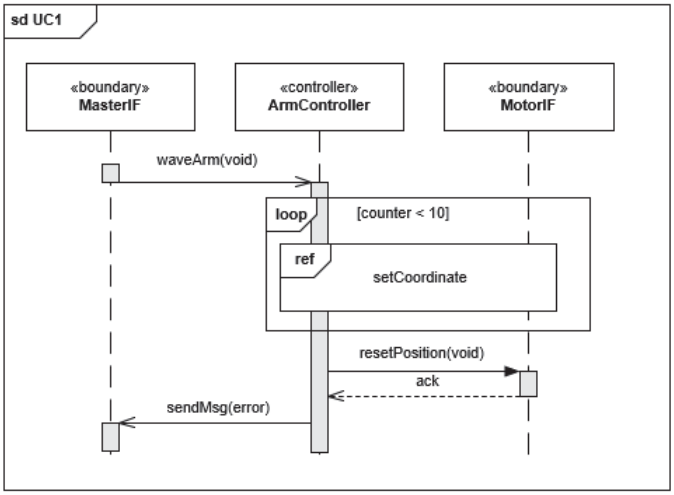
Med udgangspunkt i Use Case 1 og det overordnede IBD på Figur 7 er der fundet følgende konceptuelle moduler:



Figur 13: Klassediagram for ArmApp Use Case 1

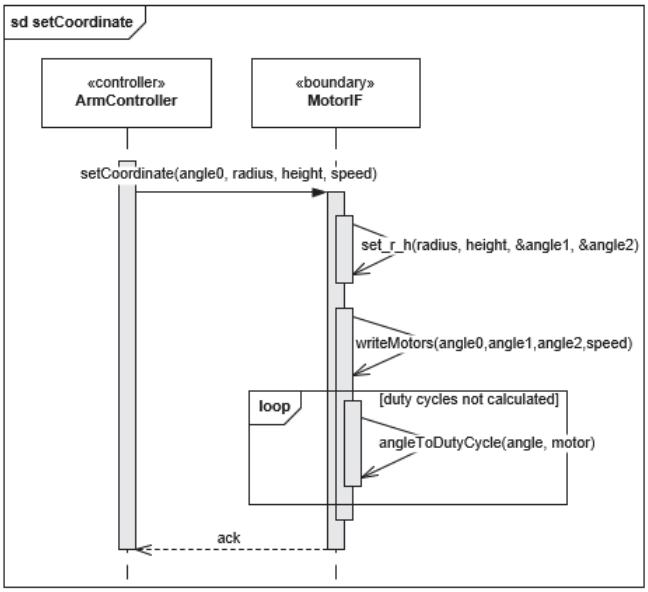
MasterIF skal modtage beskeder fra Master og fortolke data modtaget derfra. MotorIF holder styr på armens position og sørger for at armens motorer har de rigtige vinkler. ArmController skal sørge for, at de rigtige funktioner bliver kaldet i forbindelse med Use Case 1.

På Figur 14 er der et sekvensdiagram, som viser funktionskald mellem modulerne i applikationen. For overskuelighed er der indsat en reference til et andet sekvensdiagram setCoordinate på Figur 15.



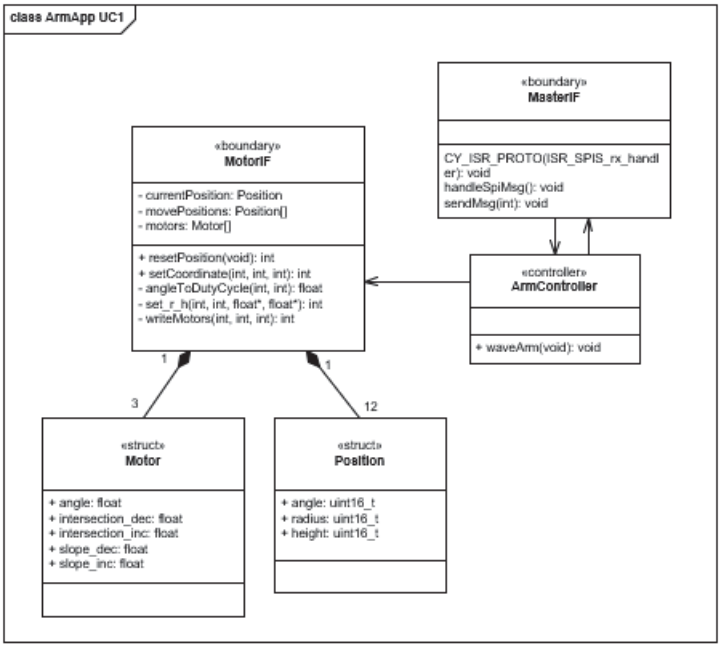
Figur 14: Sekvensdiagram for ArmApp Use Case 1

Controller-modulet kalder setCoordinate() et antal gange på en måde så det ligner, at armen vinker. Når den er færdig kaldes resetPosition(), så armen stilles i en default position og er klar til at spille. På næste figur vises det, hvilke funktioner der kaldes internt i MotorIF for at bevæge armen. En beskrivelse af funktionerne findes i afsnit 5.3.



Figur 15: Sekvensdiagram for kald af setCoordinate()

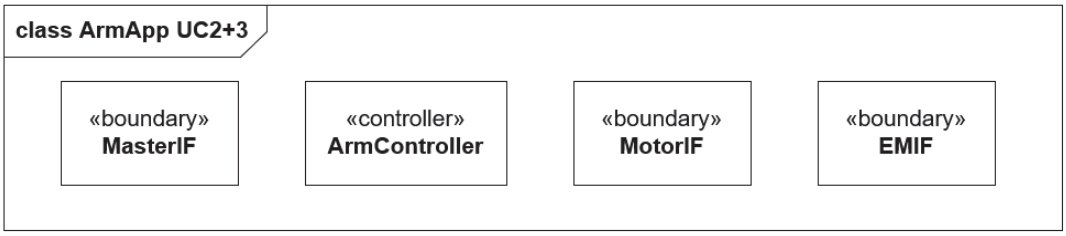
På Figur 16 er klassediagrammet opdateret med funktioner og vigtige attributter. Da koden er implementeret i *c*, er private funktioner og attributter gjort private ved at erklære dem static. Dermed kan de ikke tilgås af andre moduler.
I forhold til det forrige klassediagram er der også tilføjet to struct-datatyper, *Motor* og *Position*, til at holder styr på information. I modsætning til konventionelle klasser har de ingen metoder, og de har ikke deres egne header og source filer. De er dog stadig taget med i klassediagrammet, fordi de er med til at give overblik over, hvilken information, der ligger i MotorIF. En nærmere beskrivelse af deres anvendelse findes i afsnit 5.3.



Figur 16: Klassediagram for ArmApp med members for Use Case 1

Use Case 2+3

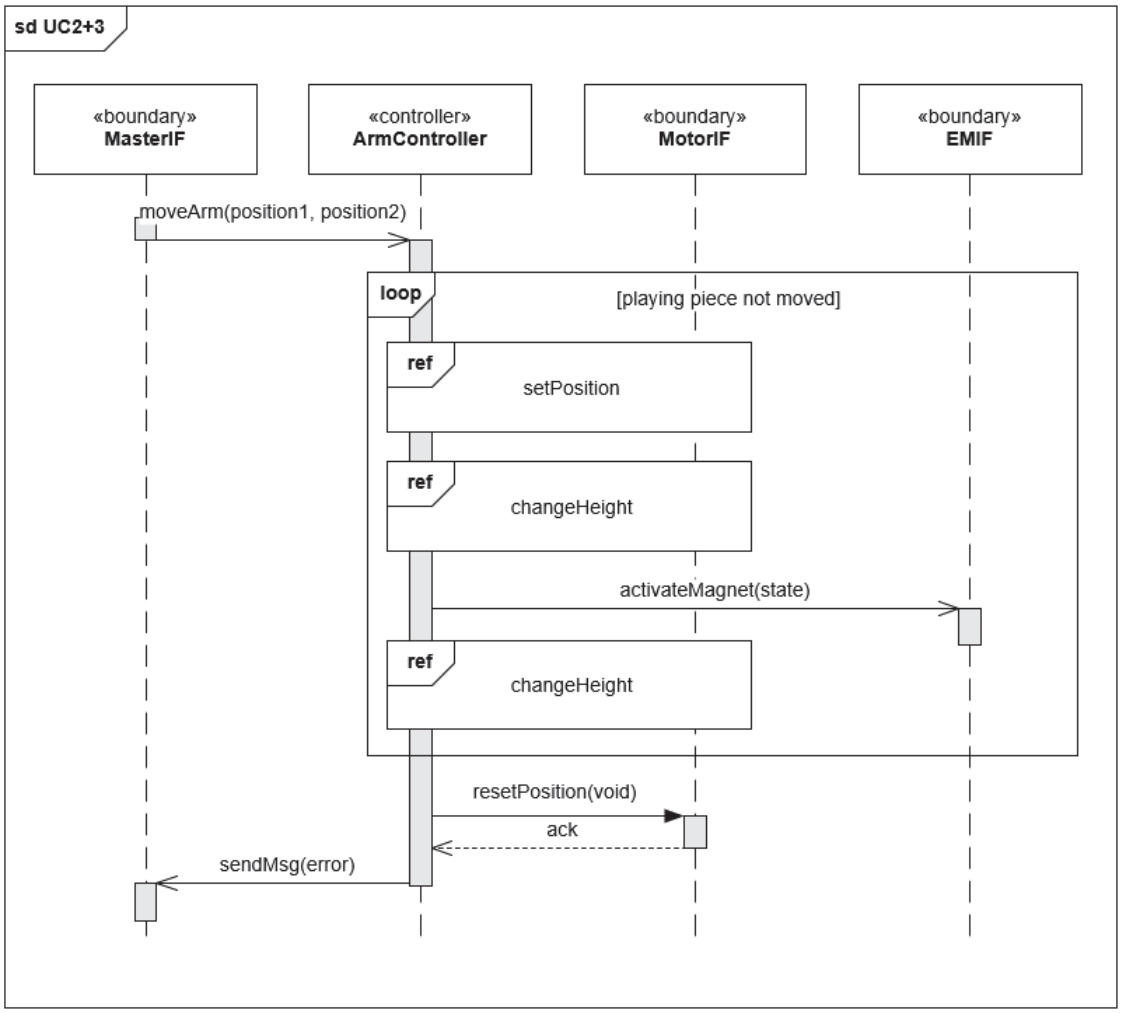
På samme måde som ved Use Case 1 findes der konceptuelle moduler på baggrund af den aktuelle Use Case og det overordnede IBD på Figur 7. Fra armens synspunkt er scenariet det samme om det er Use Case 2 eller 3, der er igang. Derfor er disse Use Cases samlet. De følgende konceptuelle moduler er fundet:



Figur 17: Klassediagram for ArmApp Use Case 2 og 3

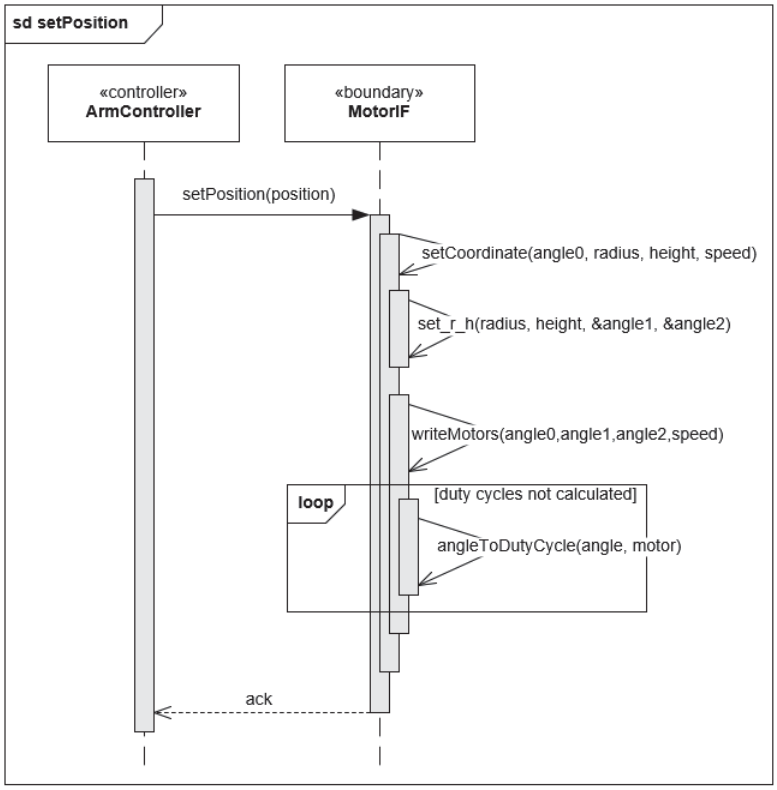
EMIF er et modul, som sørger for at aktivere og deaktivere gribemekanismen (elektromagneten) på armen. De øvrige moduler har de samme opgaver som i Use Case 1. Nu er det blot andre funktioner der kaldes mellem modulerne.

På Figur 18 er der et sekvensdiagram, som viser funktionskald mellem modulerne i applikationen. For at overskueligøre og undgå gentagelser i diagrammet er der tilføjet en reference til sekvensdiagrammet setPosition på Figur 19 og til changeHeight på Figur 20.



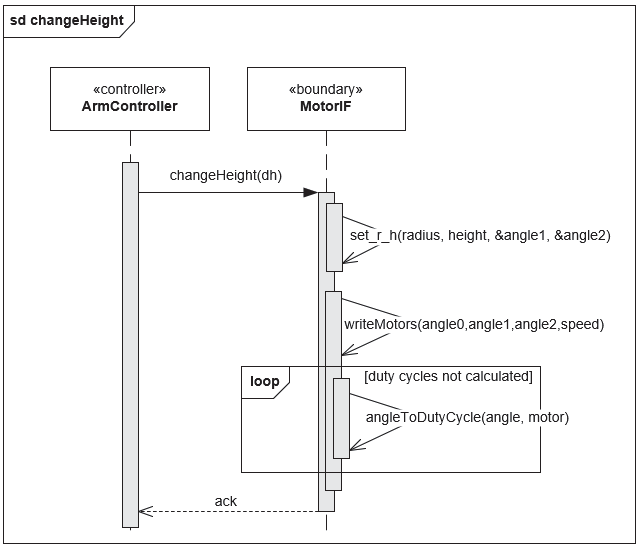
Figur 18: Sekvensdiagram for ArmApp Use Case 2 og 3

De enkelte funktioner er beskrevet i afsnit 5.3. Princippet i Use Case 2 og 3 er, at armen sættes i en bestemt position et stykke over den brik, der skal løftes. Dernæst sænkes armen med changeHeight(), elektromagneten aktiveres for at samle brikken op, og armen løftes igen. Dette gentages igen, men nu deaktiveres elektromagneten i stedet, så brikken slippes. Til sidst kaldes resetPosition(), så armen er klar til næste træk.



Figur 19: Sekvensdiagram for kald af setPosition()

setPosition() anvender setCoordinate, som også blev brugt i Use Case 1.

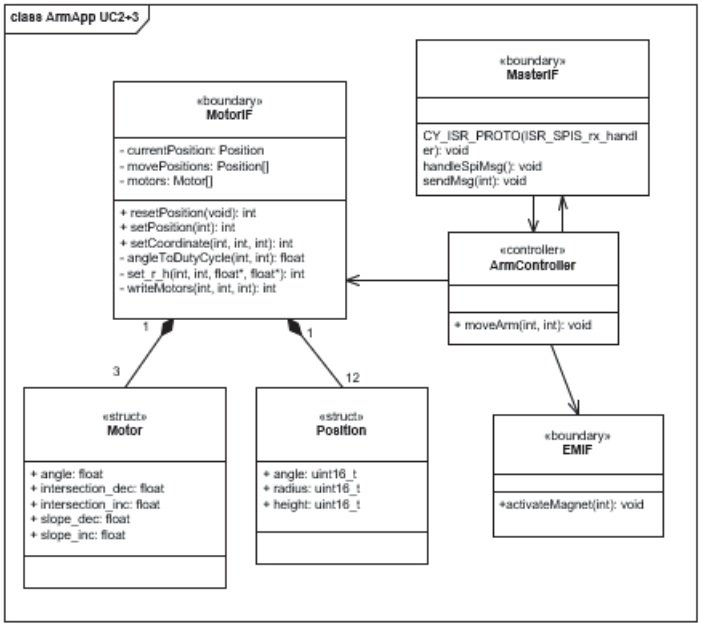


Figur 20: Sekvensdiagram for kald af changeHeight()

Sekvensdiagrammet for changeHeight() minder om sekvensdiagrammet for setCoordinate() på Figur 15. Forskellen

på de to funktioner er, at changeHeight() tager udgangspunkt i armens nuværende position og kun ændrer højden. setCoordinate() ændrer både vinkel, radius og højde uafhængigt af armens nuværende position.

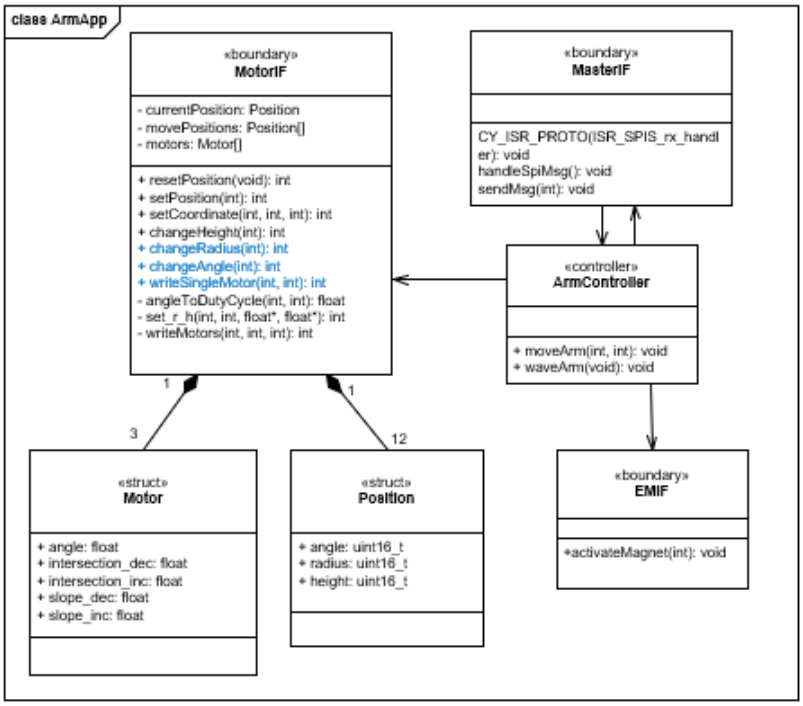
På Figur 21 er klassediagrammet opdateret med funktioner og vigtige attributter. Da koden er implementeret i *c*, er private funktioner og attributter gjort private ved at erklære dem static. Dermed kan de ikke tilgås af andre moduler.
I forhold til det forrige klassediagram er der også tilføjet to struct-datatyper, *Motor* og *Position*, til at holde styr på information. En nærmere beskrivelse af deres anvendelse findes i afsnit 5.3.



Figur 21: Klassediagram for ArmApp med members for Use Case 1

Samlet klassediagram for ArmApp

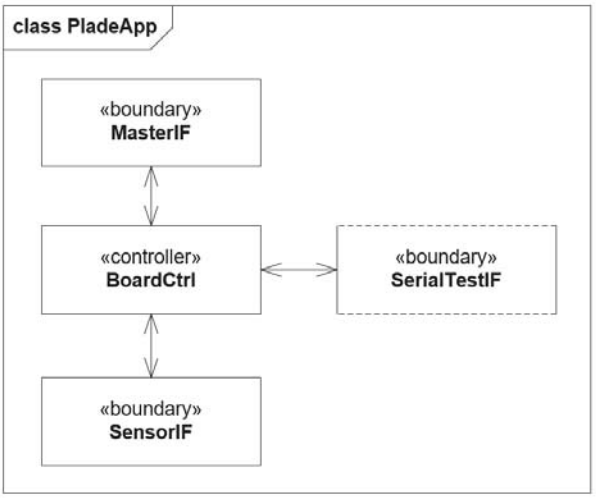
På Figur 22 ses et samlet klassediagram for ArmApp, som indeholder funktioner fra alle Use Cases. Der er tilføjet tre ekstra funktioner i MotorIF, som er markeret med blåt. Disse funktioner anvendes ikke i nogen Use Cases, men er primært implementeret med henblik på debugging og videre arbejde.



Figur 22: Samlet klassediagram for armApp

3.3 Applikationsmodel PladeApp

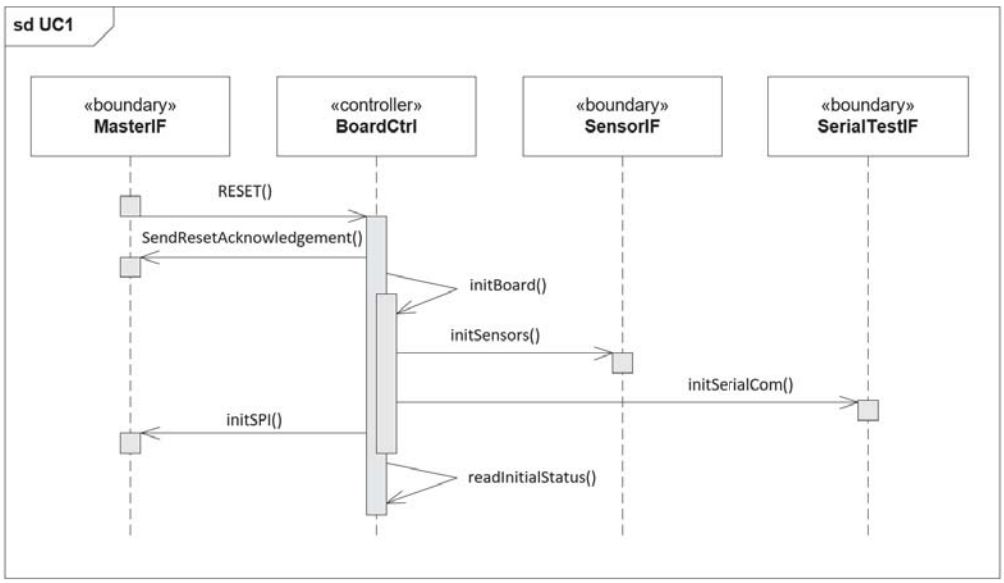
På Figur 23 ses det indledende klassediagram for plade-applikationen. Da softwaren skrives i ren c implementeres applikationen ved brug af moduler fremfor klasser. Boundary klasserne **MasterIF** og **SensorIF** håndterer de grænseflader som plade-applikationen, jævnfør ibderne på Figur 7 og Figur 9, skal kunne kommunikere med. Af hensyn til debugging og test plade-applikationen er der yderligere tilføjet en boundary-klasse **SerialTestIF**, som er benyttet til at simulere metodekald fra masteren under softwareudviklingen. Da UC1 udelukkende står for initiering af spillepladen og da metoderne, som skal benyttes i UC2+3, ligner hinanden meget, er det besluttet, at Use Case controller klasserne for de tre Use Cases samles i en fælles controller-klasse som døbes **BoardCtrl**.



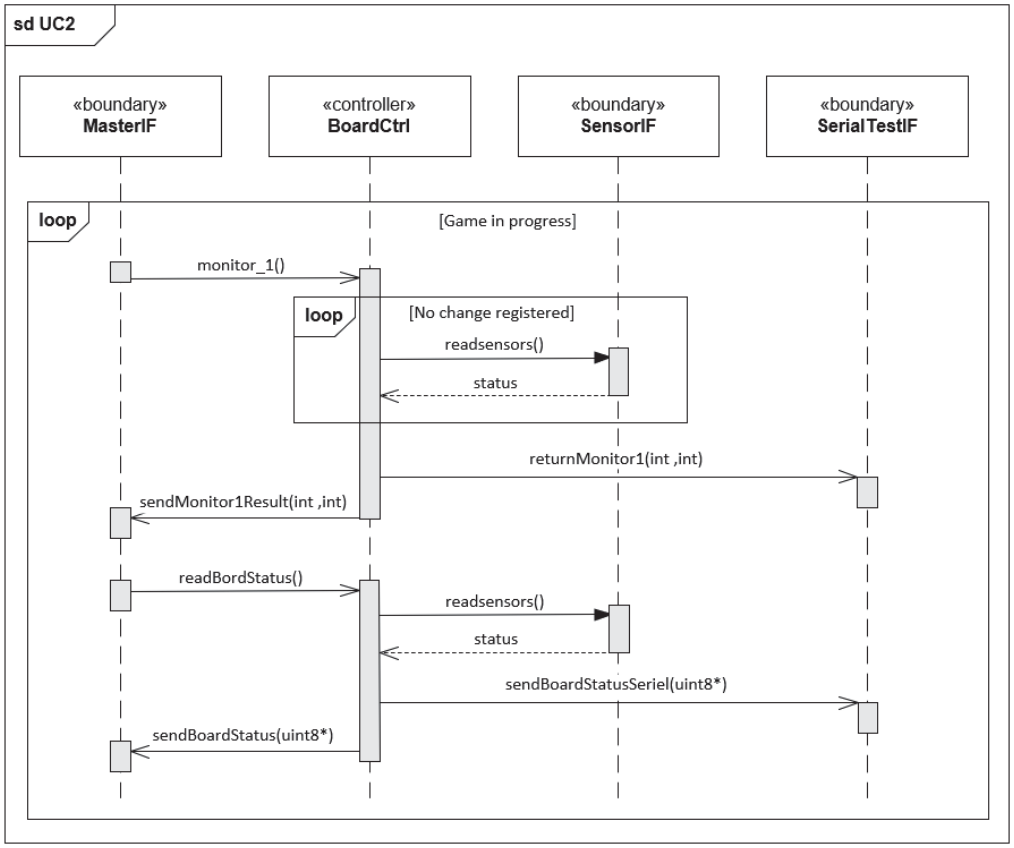
Figur 23: Indledende klassediagram for PladeApp

Sekvensdiagrammer:

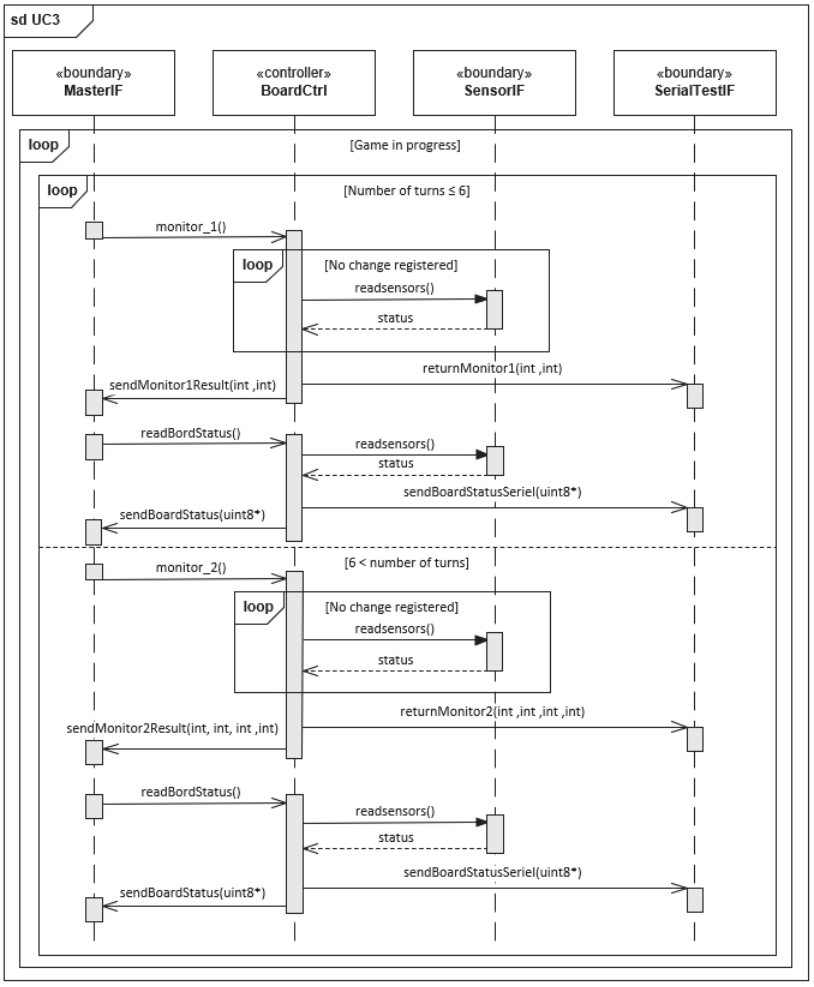
I det følgende afsnit introduceres sekvensdiagrammer for afviklingen af systemes Use Cases på plade-applikationen. I sekvensdiagrammerne bør det bemærkes, at aflæsningen og monitorering af sensorerne (readBoardStatus, monitor1 og monitor2) reelt set er mere kompliceret end hvad der præsenteres her. Særligt monitor-funktionerne er relativt komplicerede og kan ikke på hensigtsmæssig vis inkluderes i de følgende sekvensdiagrammet. Derfor introduceres en forsimplet pseudo-funktion readSensors() til brug i sekvensdiagrammerne. For forklaring af de egentlige algoritmer henvises til afsnit 7.



Figur 24: Sekvensdiagram for UC1

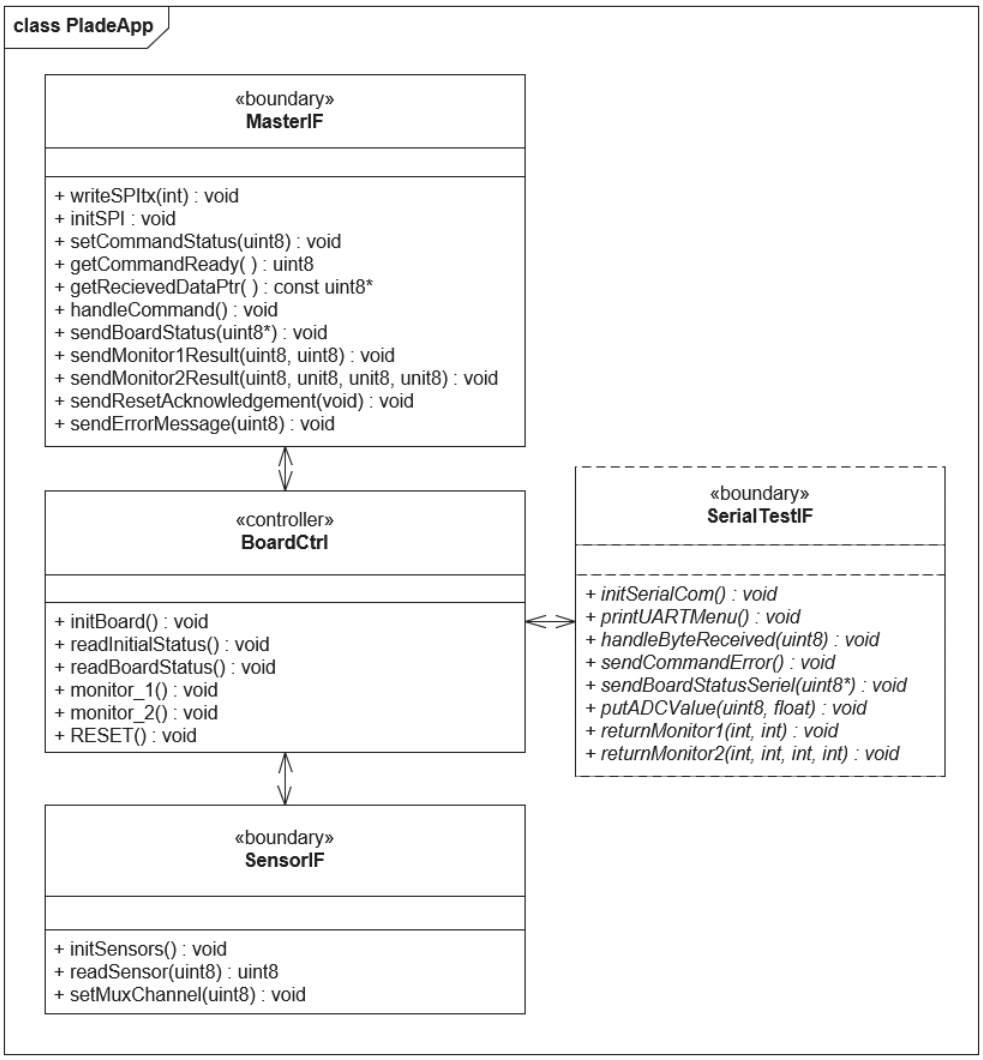


Figur 25: Sekvensdiagram for UC2



Figur 26: Sekvensdiagram for UC3

Klassediagram med metoder:



Figur 27: Klassediamgram opdateret med metoder

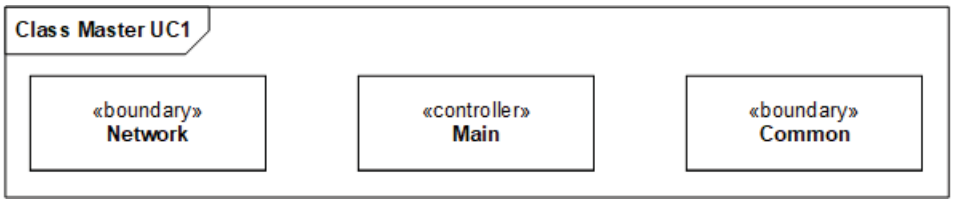
3.4 Applikationsmodel MasterApp

I dette afsnit vil MasterApp blive beskrevet. Diagrammerne er delt op i de 3 Use Cases og hvert afsnit vises der et klasse diagram og sekvensdiagrammerne for denne Use Case. Til sidst vil der komme et overordnet sekvensdiagram.

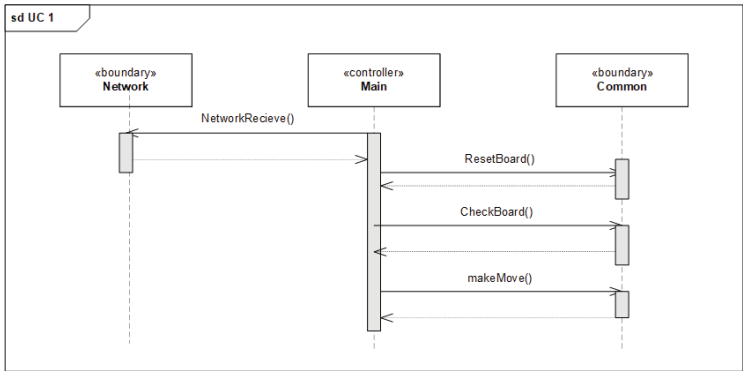
Use Case 1: Initere spil

På Figur 28 kan man se vores klassediagram for Use Case 1. Klassen Network fungerer som kommunikation med vores GUIrpi, klassen Common fungerer som kommunikation til spillepladen og armen, og som en fælles klasse som Finite og Infinite klasserne arver fra. I denne Use Case skal spillet initeres, og dette differe ikke for hvilken mode man spiller. Sekvensdiagrammet for Use Case 1 kan se på Figur 29. Det vigtige at lægge mærke til i dette sekvensdiagram er, at der modtages data fra Network til at starte med. Dette data er spilmode, sværhedsgrad

og spilstarter. Herefter bliver der tjekket boardet med et reset på boardet for bagefter at tjekke om boardet er klar. Til sidst skal armen vinke.



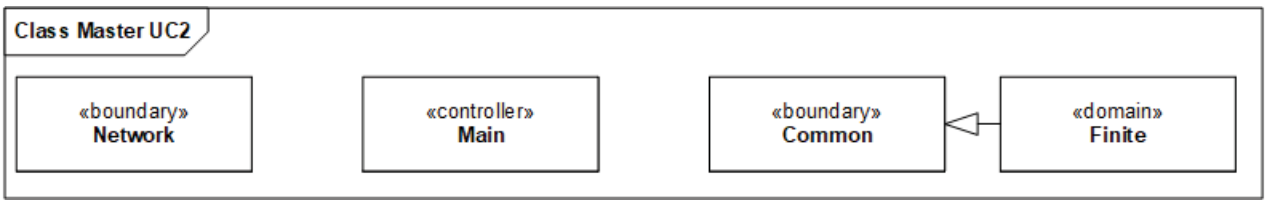
Figur 28: Klassediagram for Use Case 1



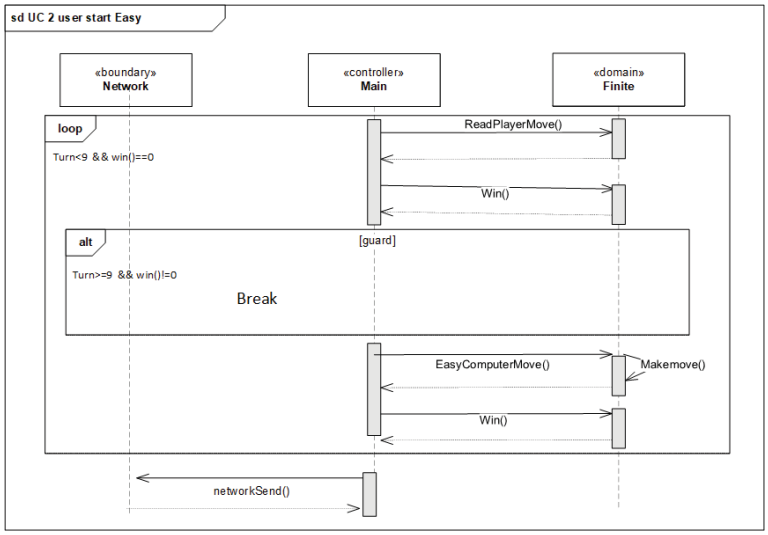
Figur 29: Sekvensdiagram for Use Case 1

Use Case 2: Finite Mode

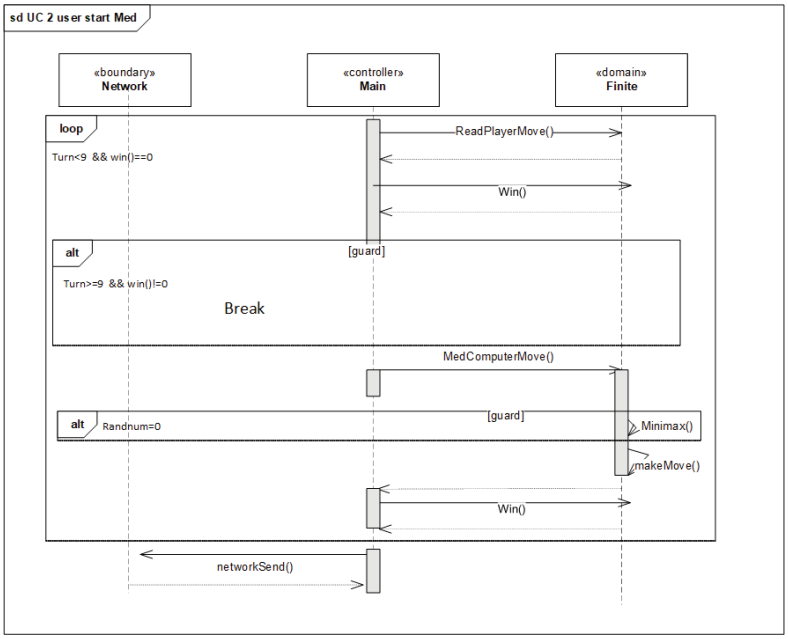
I Finite mode har vi tilføjet en klasse som skal beregne computerens træk ud fra sværhedsgraden. Klassediagrammet for denne Use Case kan ses på Figur 30. Der er lavet 6 sekvensdiagrammer, da der er to mulige spilstartere og tre sværhedsgrader, disse kan ses fra Figur 31 til Figur 36. Her skal man lægge mærke til, at der bliver tjekket om turn er over 9, som betyder, at boardet er fuldt, samtidig bliver der tjekket om en spiller har vundet efter hvert træk. Efter dette bliver hvem der vandt sendt til klassen Network som for at sende dette videre. Forskellen på hvem der starter og hvad sværhedsgraden er, betyder ikke meget for vores sekvensdiagrammer, forskellen ligger i de funktioner, der bliver kaldt, men sekvensdiagrammerne er alligevel taget med.



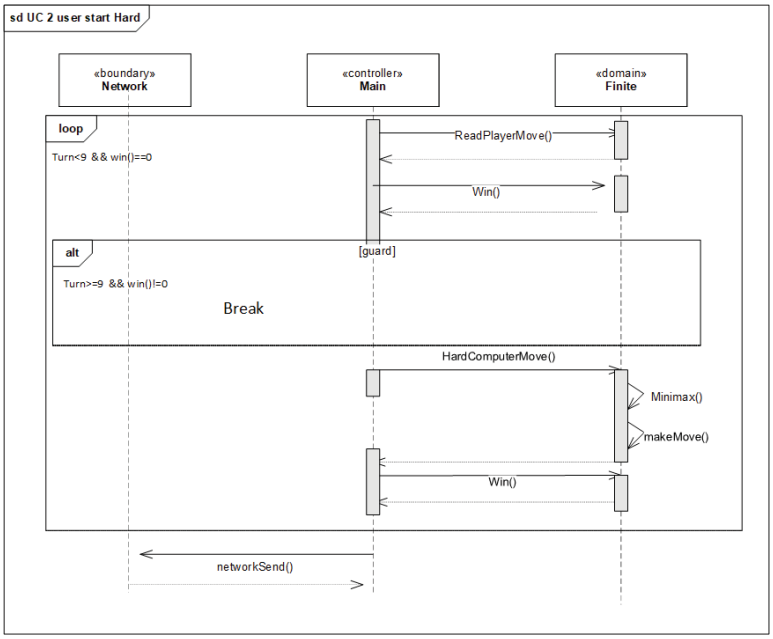
Figur 30: Klassediagram for Use Case 2



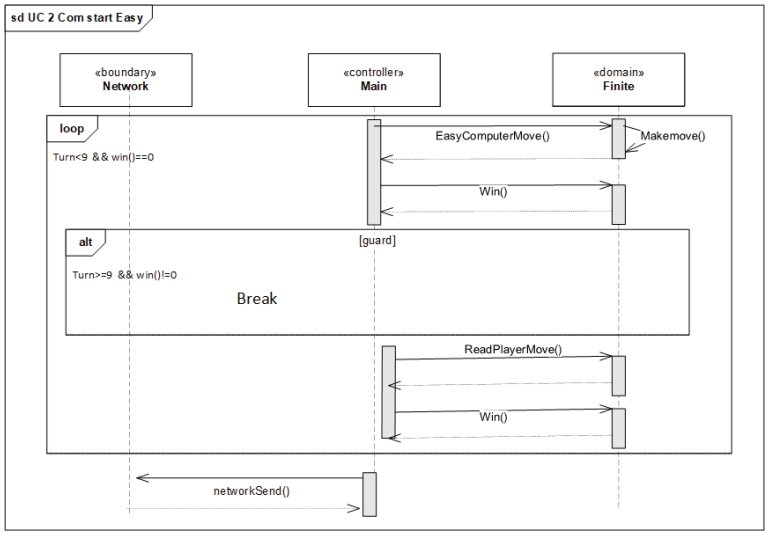
Figur 31: Sekvensdiagram for Use Case 2 bruger er spilstarter, sværhedsgrad Easy



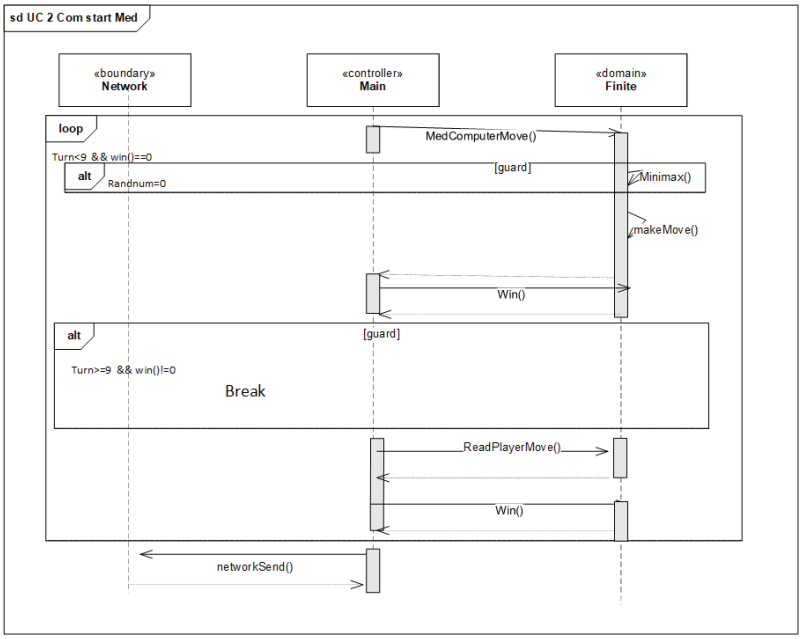
Figur 32: Sekvensdiagram for Use Case 2 bruger er spilstarter, sværhedsgrad Medium



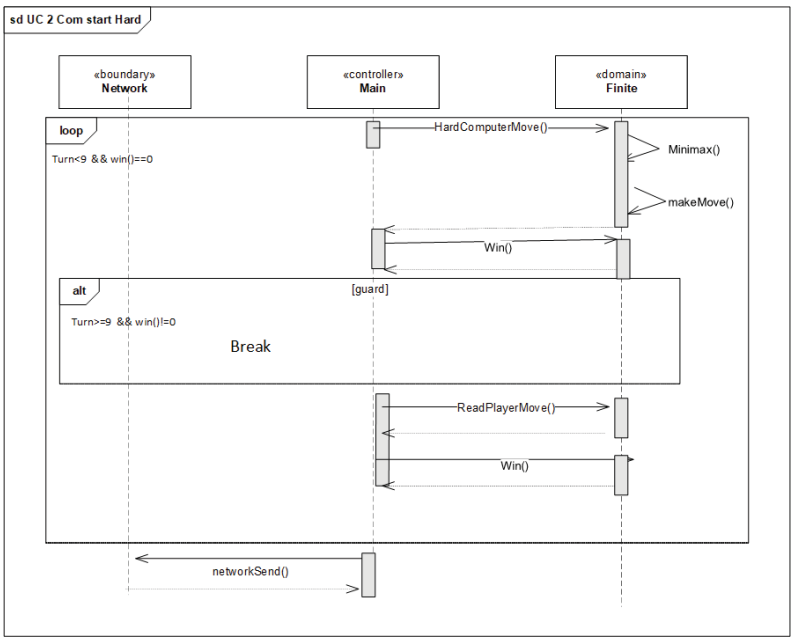
Figur 33: Sekvensdiagram for Use Case 2 bruger er spilstarter, sværhedsgrad Hard



Figur 34: Sekvensdiagram for Use Case 2 TTT er spilstarter, sværhedsgrad Easy



Figur 35: Sekvensdiagram for Use Case 2 bruger er spilstarter, sværhedsgrad Medium

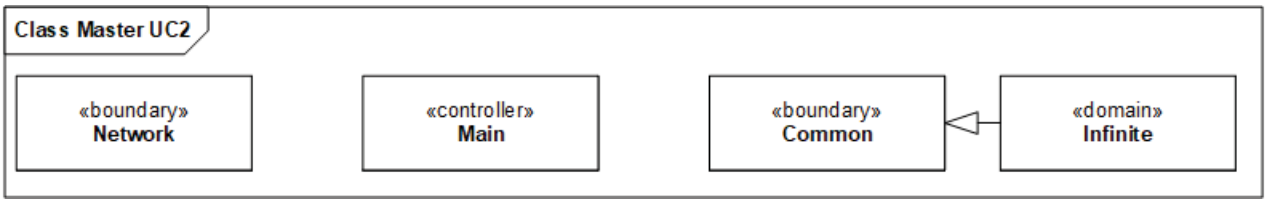


Figur 36: Sekvensdiagram for Use Case 2 bruger er spilstarter, sværhedsgrad Hard

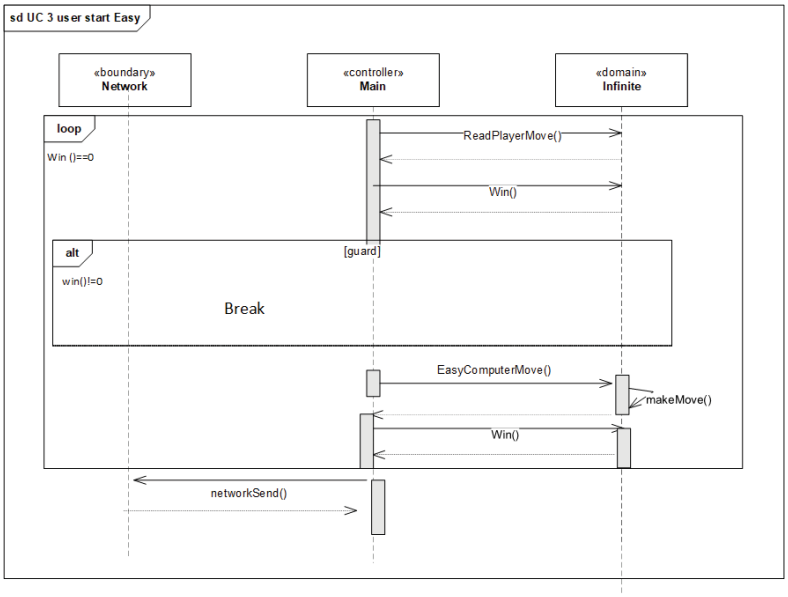
Use Case 3: Infinite Mode

I Infinite mode har vi tilføjet en klasse som skal beregne computerens træk ud fra sværhedsgraden for Infinite mode. Klassediagrammet for denne Use Case kan ses på Figur 37. Der er lavet 6 sekvensdiagrammer, da der er to mulige spilstartere og tre sværhedsgrader, disse kan ses fra Figur 38 til Figur 43. Her skal man lægge mærke til, at turn ikke længere bliver tjekket. Det er, fordi IM skal kunne spille indtil en vinder er fundet, så det

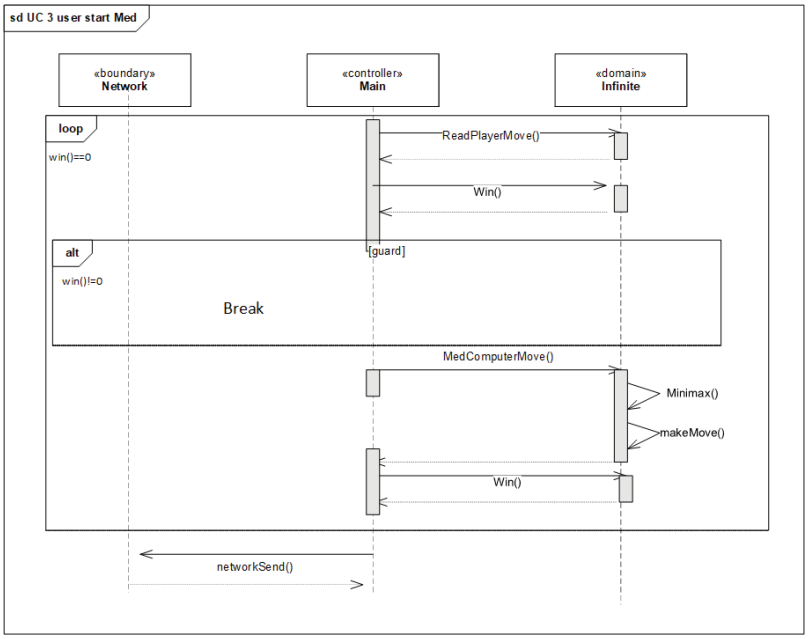
giver ikke mening, at turn skal tjekkes længere. Forskellen mellem hvem der starter og hvad sværhedsgraden er, betyder endnu engang ikke meget for vores sekvensdiagrammer.



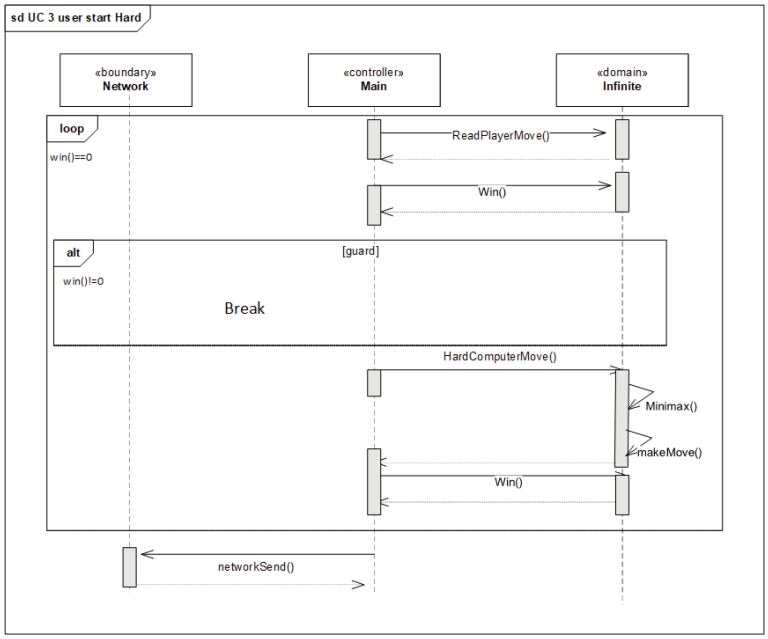
Figur 37: Klassediagram for Use Case 3



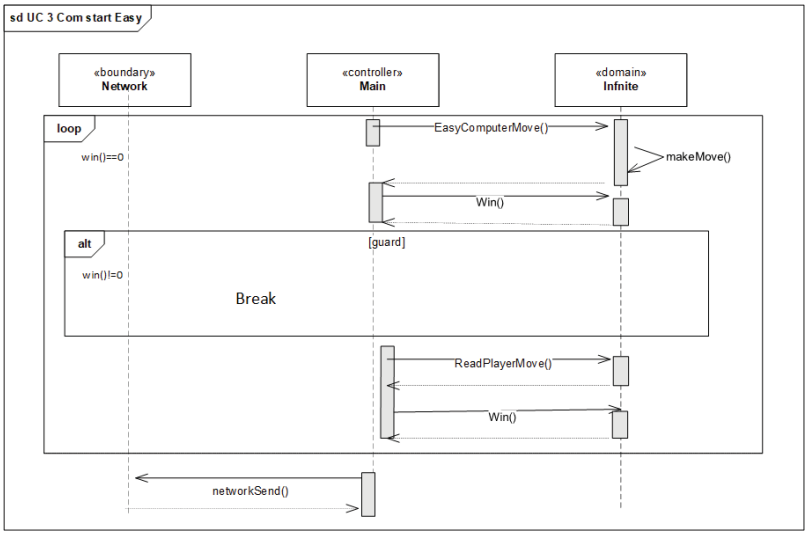
Figur 38: Sekvensdiagram for Use Case 3 bruger er spilstarter, sværhedsgrad Easy



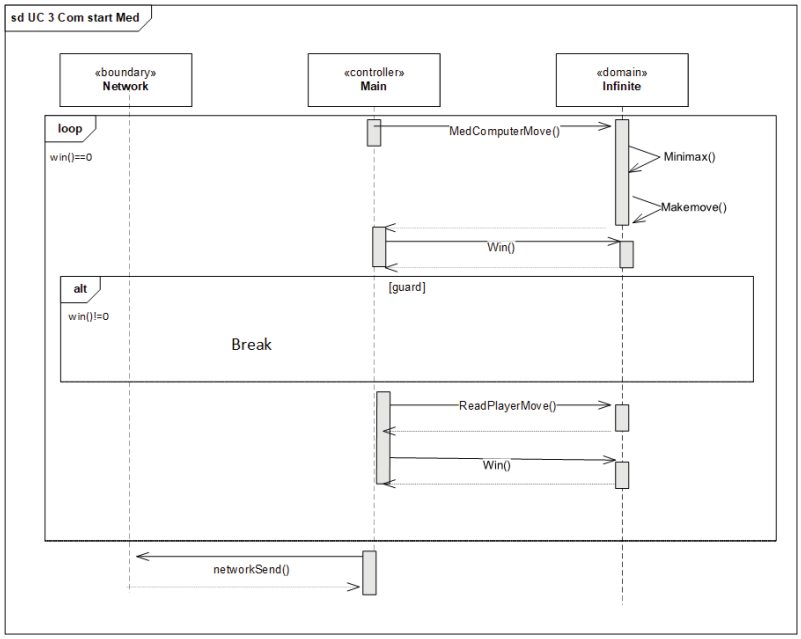
Figur 39: Sekvensdiagram for Use Case 3 bruger er spilstarter, sværhedsgrad Medium



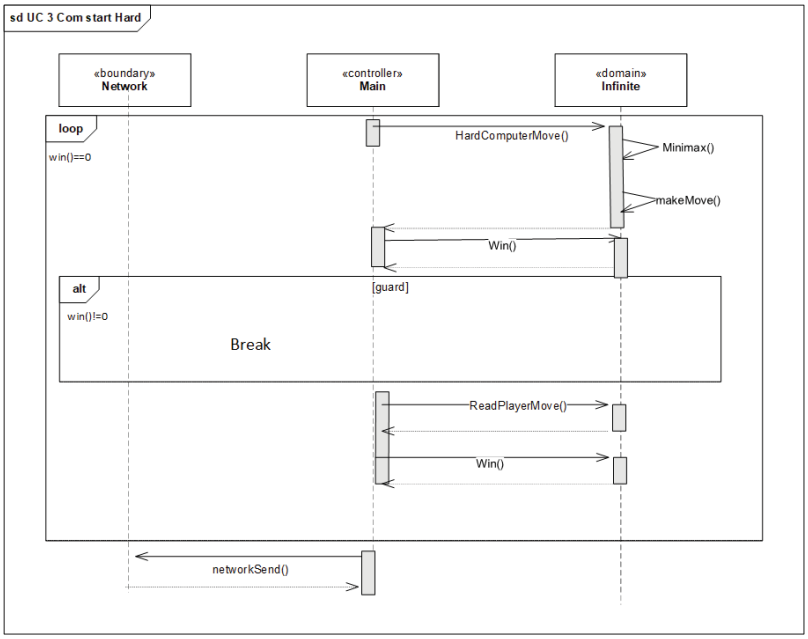
Figur 40: Sekvensdiagram for Use Case 3 bruger er spilstarter, sværhedsgrad Hard



Figur 41: Sekvensdiagram for Use Case 3 TTT er spilstarter, sværhedsgrad Easy



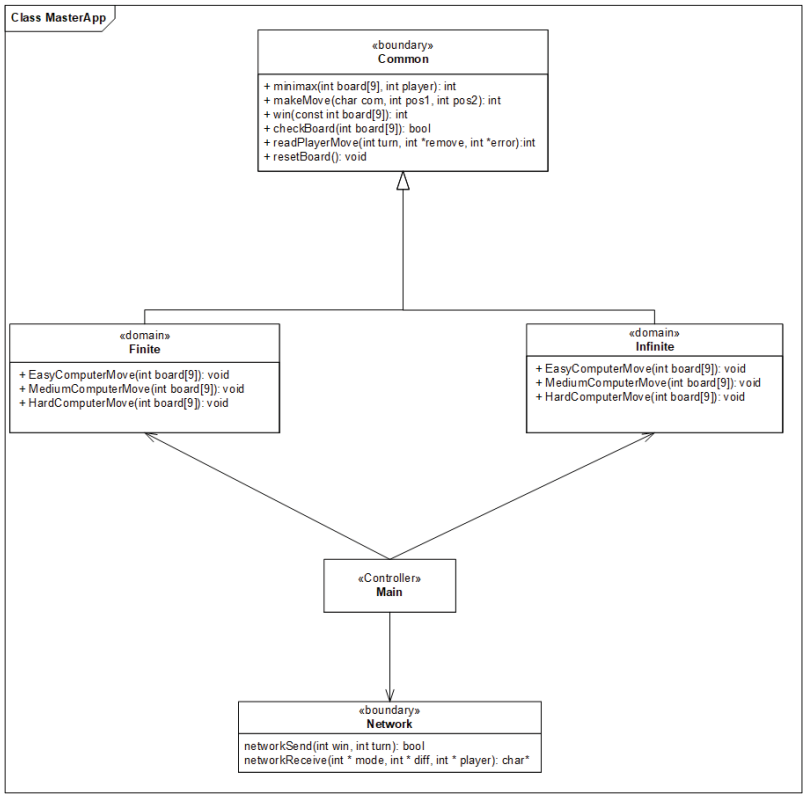
Figur 42: Sekvensdiagram for Use Case 3 bruger er spilstarter, sværhedsgrad Medium



Figur 43: Sekvensdiagram for Use Case 3 bruger er spilstarter, sværhedsgrad Hard

Opdateret Klassediagram

Der kan ud fra sekvensdiagrammerne findes funktioner for vores klasser. Det opdaterede applikationsmodel kan findes på Figur 44.



Figur 44: Opdateret KlasseDiagram

3.5 Applikationsmodel InterfaceApp

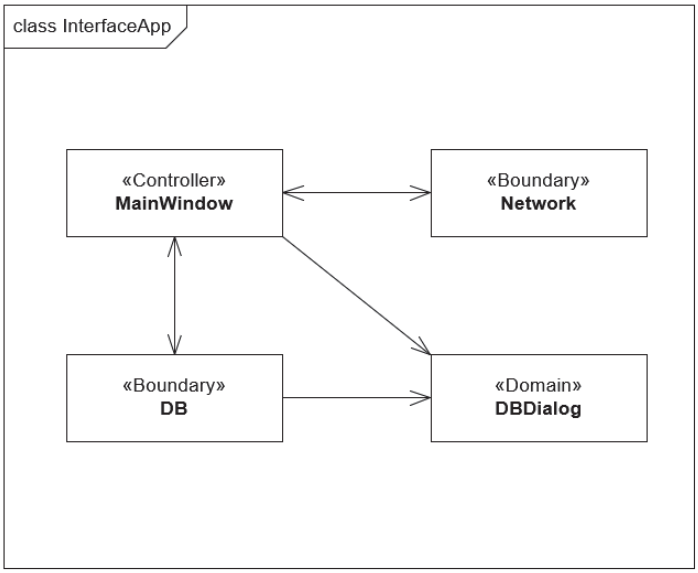
Dette afsnit omhandler InterfaceAppens softwarearkitektur. Der vil i denne forbindelse blive opstillet et klassediagram for dette modul. Herefter vil der blive opstillet sekvensdiagrammer for hver Use Case, med henblik på InterfaceAppen. Når disse er gennemløbet, er der bedre mulighed for at opdatere klassediagrammet med private og public members.

Klassediagram for InterfaceApp:

I Figur 45 ses det opstillede klassediagram for softwaren bag InterfaceApp.

Heri består en controller-klasse, som holder styr på hele systemet, altså MainWindow. Denne er også vinduet der bliver vist på touchskærmen, og hvori størstedelen af det anvendelsesmessige i forhold til brugeren af systemet ligger. Denne har en tovejskommunikation mellem begge boundry-klasser, DB og Network. Network er det trådløse bindeled mellem GUI og Master, en Wi-Fi forbindelse igennem en socket. DB sørger for at oprette forbindelse til databasen som indeholder highscore, og manipulere med dataen heri.

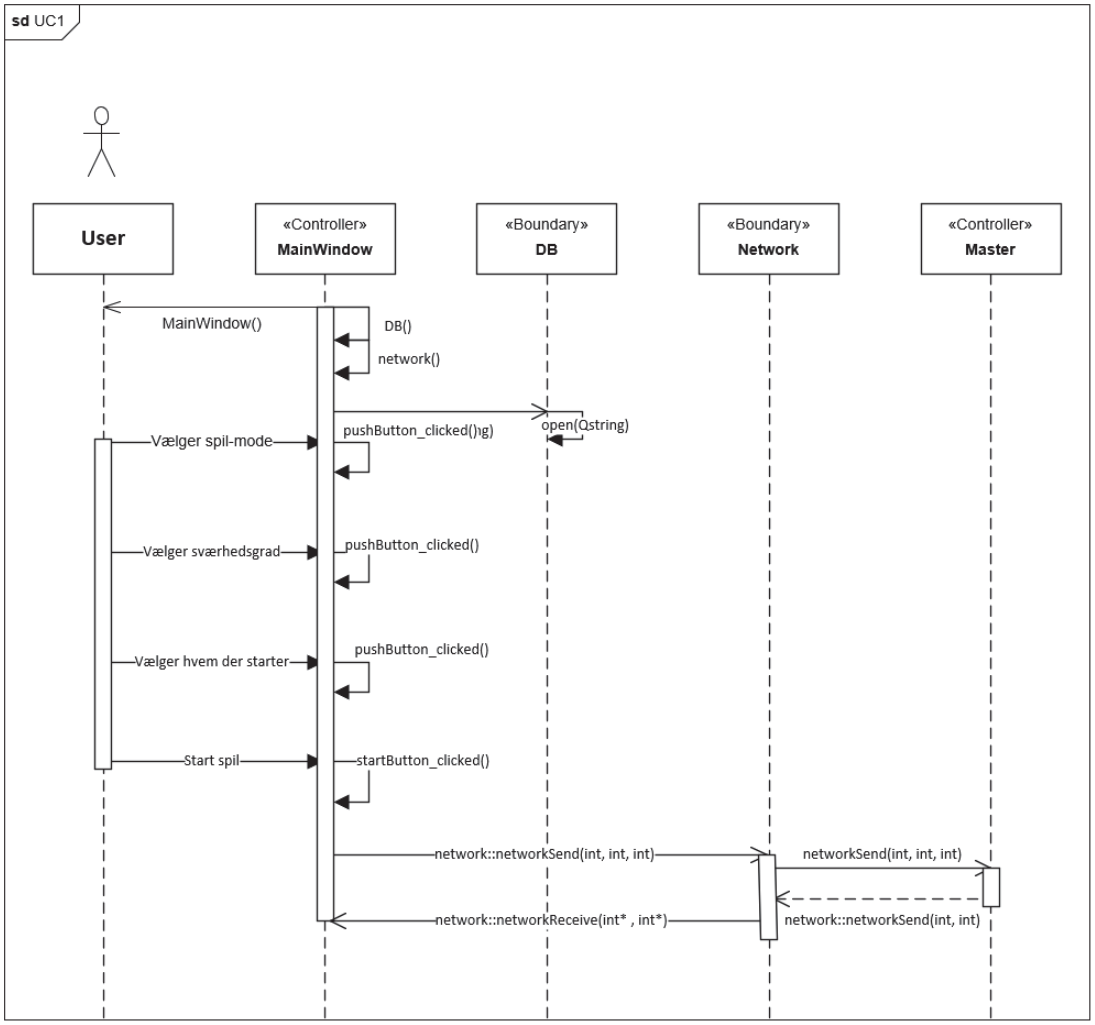
DBDialog er et vindue for sig selv, der kan åbnes af MainWindow. Heri vises highscoren, og disse datasæt hentes fra en envejkommunikation fra DB.



Figur 45: Klassediagram for InterfaceApp

SD for Use Case 1: Initier Spil:

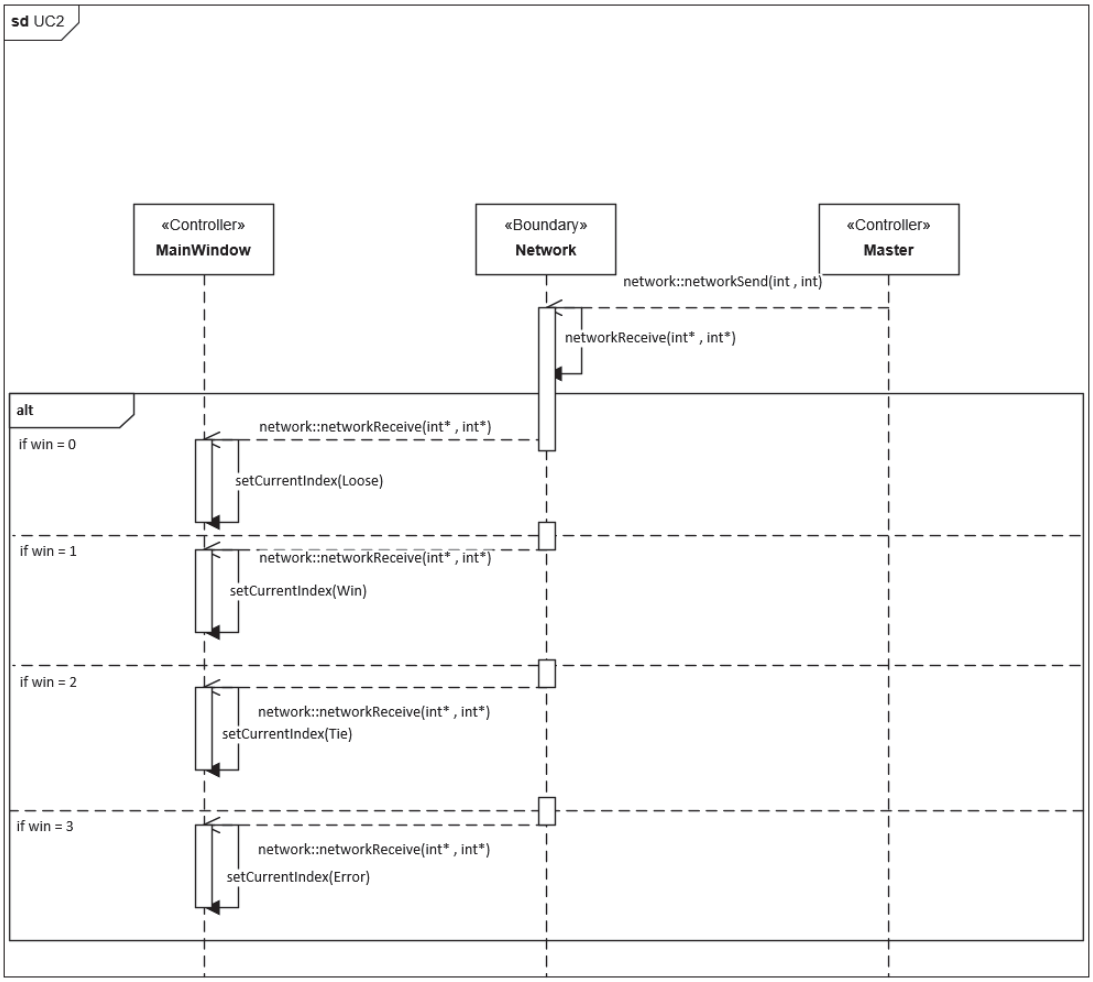
Første Use Case omhandler initieringen af et spil. Dette foregår som vist på Figur 46 ved, at MainWindow først og fremmest bliver vist til brugeren. I samme omgang bliver der oprettet en klasse til DB og Network. DB går ind og åbner en database, med samme navn som den parsede Qstring. Brugeren kan herefter vælge mode, sværhedsgrad og hvem der starter spillet på GUI. Dette gøres ved at bruge knapper, som hver fører brugeren til en ny valgmulighed. Efter at disse tre valg er blevet taget, er der mulighed for at starte/initiere spillet ved brug af en knap på GUI. Når spillet er startet, vil GUI sende valgene over netværkskommunikationen til Master, og vente på, at brugeren bliver færdig med spillet således Master sender et resultat tilbage.



Figur 46: SD for InterfaceApp - UC1

SD for Use Case 2: Finite Mode:

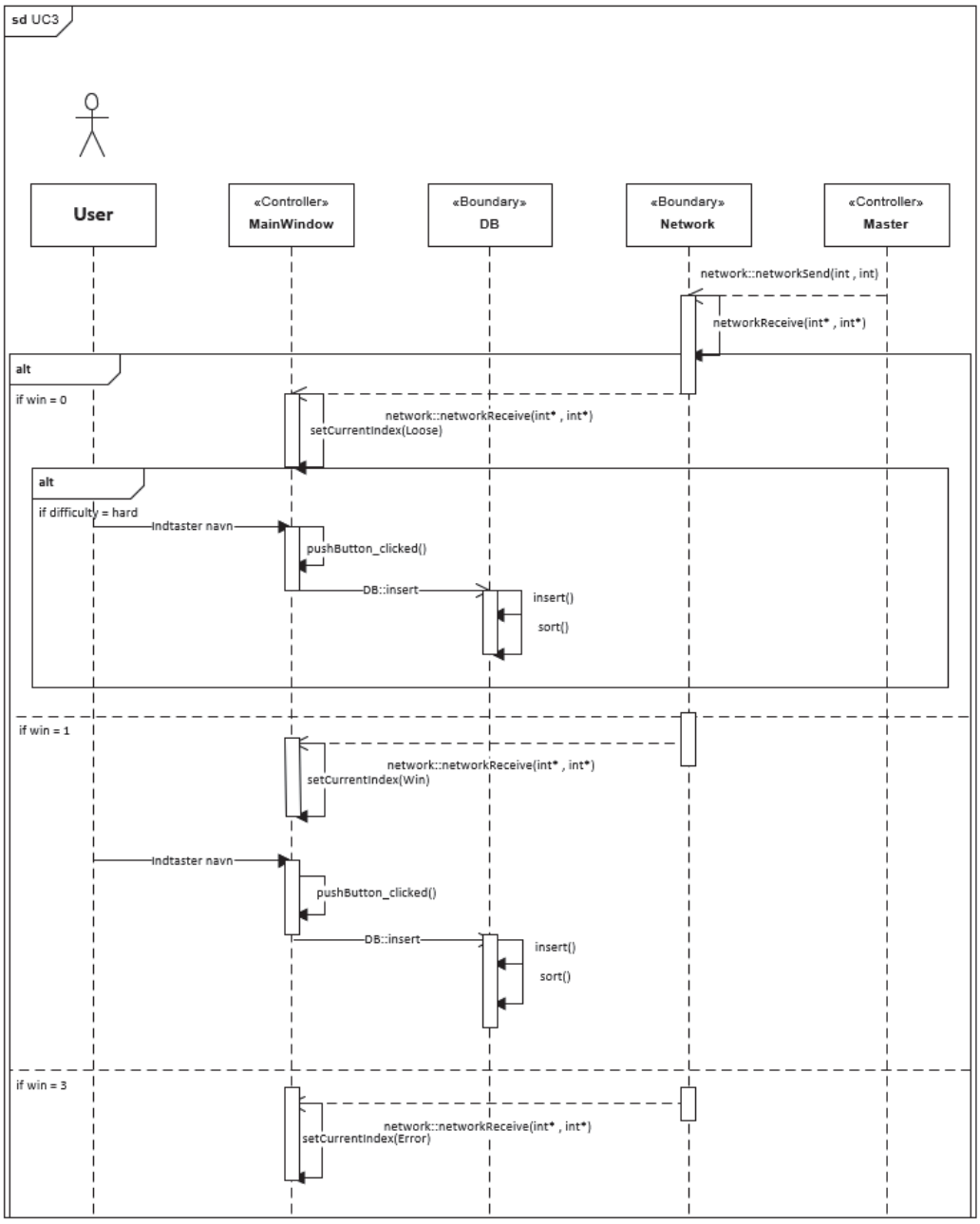
Figur 47 viser hvordan Use Case 2 forløber i forhold til GUI. Efter at spillet er spillet færdigt, vil Master sende et resultat til GUI. Den første parsede værdi vil her være win, som er bestemmende for om spilleren har tabt. Hvis win er 0, har spilleren tabt, er win 1 har spilleren vundet, er win 2 er spillet uafgjort og er win 3, er der sket en fejl. Herefter vil en MainWindow vise en passende skærm på GUI.



Figur 47: SD for InterfaceApp - UC2

SD for Use Case 3: Infinite Mode:

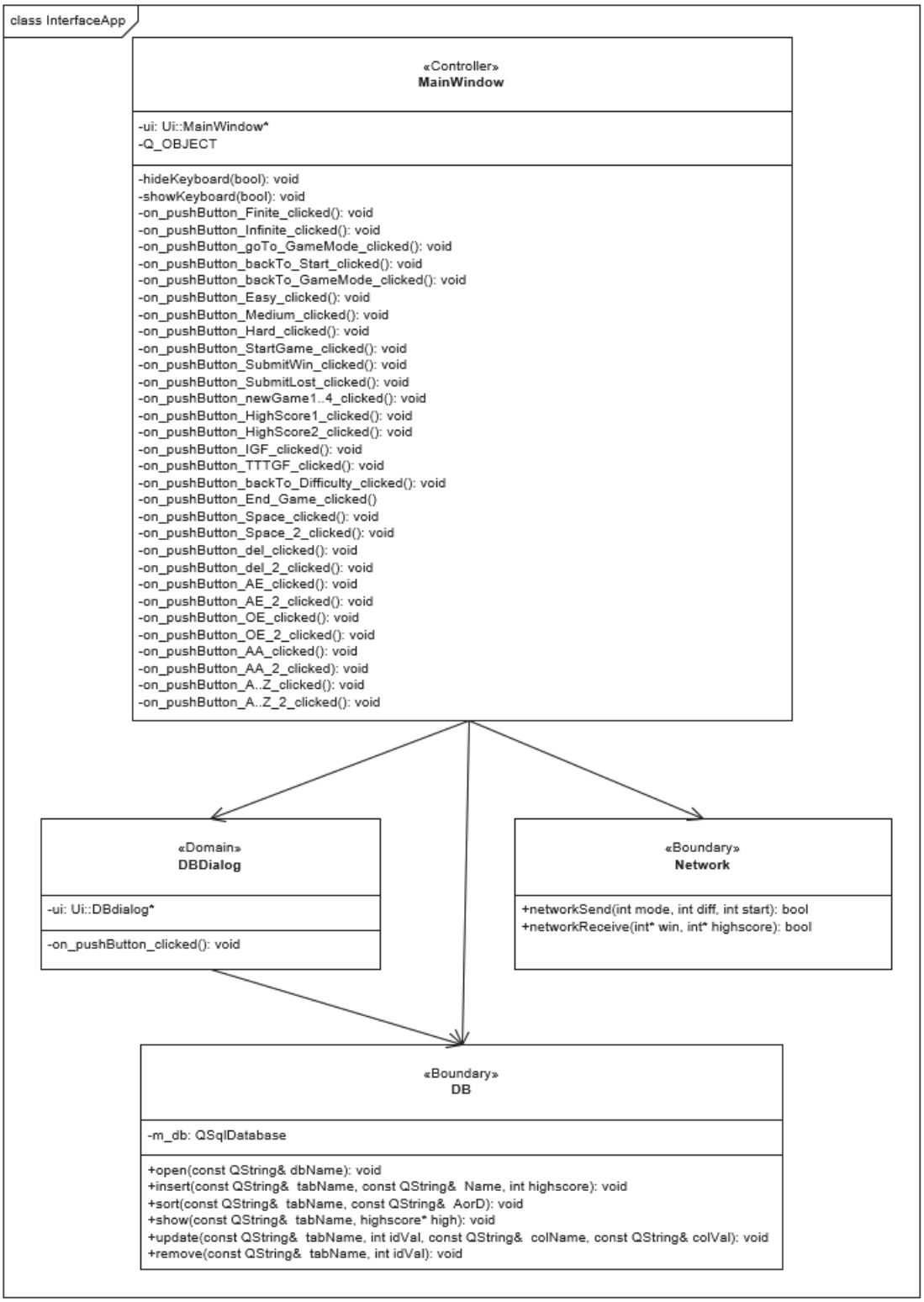
I Figur 48 er sekvensdiagrammet for Use Case 3 i forhold til GUI opstillet. Denne minder utrolig meget om Use Case 1 der er dog den undtagelse, at hvis spilleren vinder (altså hvis win er 1), får personen mulighed for at indtaste sit navn. Efter et tryk på en knap, vil dette blive sat ind i databasen, sammen med den anden parsede parameter fra networkSend fra Master, nemlig highscoren. Efter dette vil databasen sortere sig selv, således at datasættet med ID 1, vil være førstepladsen, ID 2 andenpladsen osv. Denne mulighed er dog stadig til stede hvis man taber, når sværhedsgraden er sat til Hard.



Figur 48: SD for InterfaceApp - UC3

Opdateret Klassediagram for InterfaceApp:

Efter at have gennemløbet alle funktioner er det nu muligt at opdatere klassediagrammet fra Figur 45. For at gøre diagrammet mere overskueligt er pushButtons fra A til Z og A2 til Z2 forkortet til hhv. on_pushButton_A..Z_clicked() og on_pushButton_A..Z_2clicked(). Beskrivelserne for disse er at finde i Afsnit 8. Beskrivelser for Network er at finde i Afsnit 10.2 og DB i Afsnit 9.



Figur 49: Opdateret KlasseDiagram for InterfaceApp

4 Software protokoller

Herunder er protokollerne for SPI og Wi-Fi kommunikation specificeret.

4.1 SPI protokol

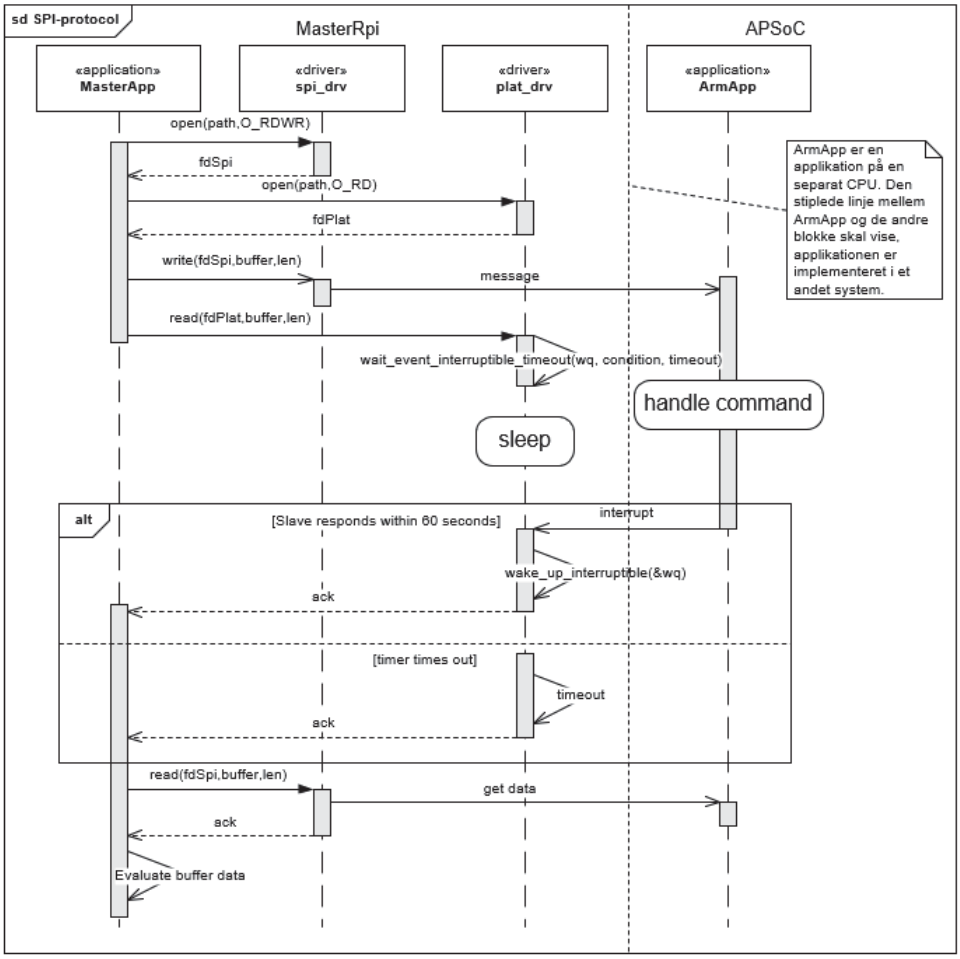
I dette afsnit beskrives SPI-protokollen, som er grænsefladen mellem Master og APSoC og PPSoC, som er SPI-slaver. I det følgende beskrives først anvendelsen af to drivere, som er lavet til at håndtere kommunikationen. Dernæst beskrives betydningen af de data, der sendes over SPI-busserne.

4.1.1 Anvendelse af drivere

For at muliggøre SPI-kommunikation mellem Master og en PSoC-slave er der lavet to linux-drivere: *spi_drv* og *plat_drv*. *spi_drv* anvendes til at sende og modtage beskeder via SPI. *plat_drv* har forskellige opgaver, men i forbindelse med SPI-protokollen er dens funktion at håndtere interrupts på to GPIO'er, en for hver slave.

I systemet er det nødvendigt at indføre interrupts i SPI-protokollen, fordi en slave ikke kan tage initiativ til kommunikation. En master tager initiativ både til at læse og skrive. Hvis en master ved, hvor lang tid det tager for slaven at klargøre sin tx-buffer, kan man i masteren indsætte et delay før der læses. Men når masteren i dette system skriver til pladen eller armen ved den ikke, hvor lang tid der går, før den kan læse. Det er afhængigt af, hvor lang tid brugeren er om at foretage et træk, og hvor langt armen skal bevæge sig. Derfor laver en slave interrupt på masteren, så den ved, hvornår den kan læse.

På Figur 50 vises et sekvensdiagram med hele sekvensen fra der er sendt en besked fra Masterapplikationen til at den modtager en returbesked fra en slave. APSoC'en er anvendt som eksempel, men princippet er nøjagtigt det samme for PPSoC'en.



Figur 50: Sekvensdiagram over anvendelse af drivere til SPI-kommunikation

På sekvensdiagrammet ses en stiplet linje mellem ArmApp og de øvrige blokke. Det skal illustrere, at ArmApp ligger på en anden CPU. Pilene til ArmApp er derfor heller ikke egentlige funktionenskald. Pilene skal blot vise, at foregår en kommunikation mellem Master og APSoC via SPI.

Først åbnes noden svarende til det SPI-device, som APSoC er koblet til. Dernæst åbnes noden svarende til det device, der håndterer interrupts fra APSoC.

MasterApp'en skriver til armen via *spi_drv*. ArmApp er nu gået i en tilstand *handle command*, hvor den udfører kommandoen fra MasterApp. Den laver et interrupt på Masteren når den er færdig. Når MasterApp læser fra *plat_drv* sover den indtil der er interrupt fra APSoC eller der er timeout.

Til sidst læses der fra APSoC.

4.1.2 Data i protokol

Start- og stop-bytes: Alle kommandoer der udveksles indenfor den serielle protokol, der er beskrevet for Armen og Spillepladen benytter ASCII '+' som startbyte og '-' som stopbyte. Start og stop-bytes benyttes til at validere længden og korrektheden af de udvekslede kommandoer.

Spilleplade:

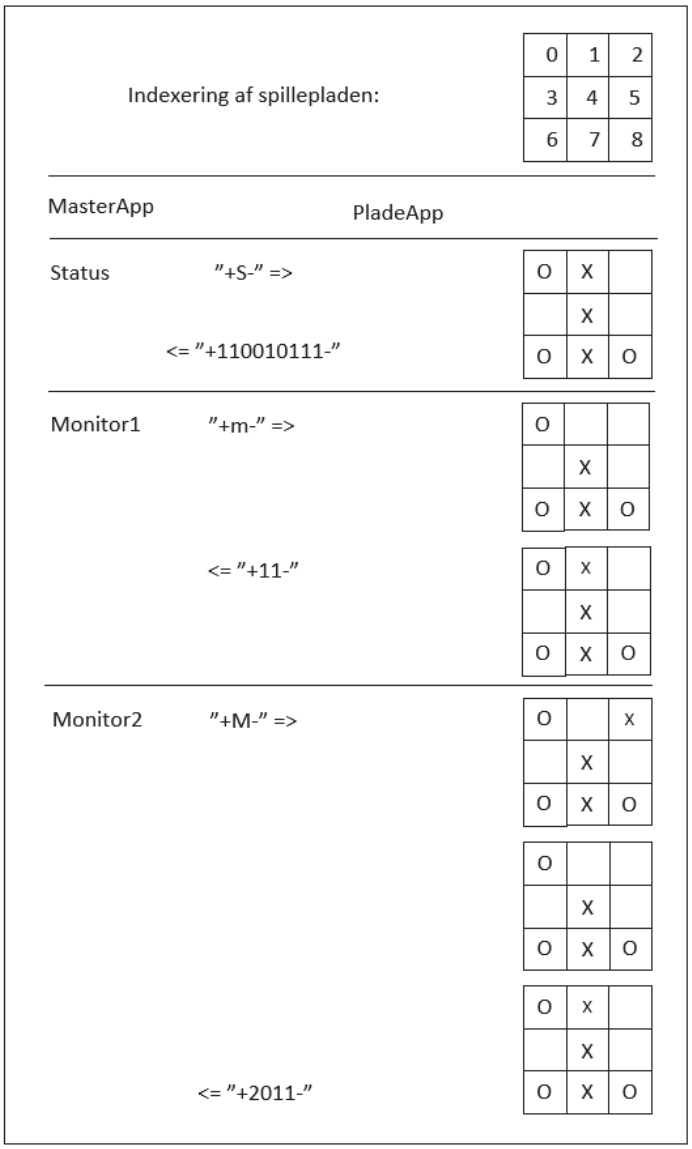
Status
Kommando-byte: 'S'
Data-bytes: ingen
Retur: Status (0/1) for alle lokationer på Spillepladen specificeret på Figur 51. Returneres i rækkefølgen lav-høj indeksering.
Beskrivelse: Kommandoen benyttes til at aflæse status for pladen.

Monitor1
Kommando-byte: 'm'
Data-bytes: ingen
Retur: Returnerer index og ny status for den første ændring der registreres på pladen i nævnt rækkefølge.
Beskrivelse: Kommandoen benyttes til at afvente brugerens træk og returnere trækket til Masteren. Monitor1 benyttes i FM og ved de første 6 runder i IM.

Monitor2
Kommando-byte: 'M'
Data-bytes: ingen
Retur: Returnerer index og ny status for den første og anden ændring der registreres på pladen i nævnt rækkefølge.
Beskrivelse: Kommandoen benyttes til at afvente brugerens træk og returnere trækket til Masteren. Monitor2 benyttes i IM efter de første 6 runder er afviklet.

RESET
Kommando-byte: 'R'
Data-bytes: ingen
Retur: 'R'
Beskrivelse: Kommandoen benyttes til at lave et software reset på PladeApplikationen. Dette kan være fordelagtigt ved start på et nyt spil, så det sikres, at PladeApplikationen ikke står og afvikler en monitor-funktion og så der indlæses en ny start-status for spillepladen.

Eksempel:



Figur 51: Seriel protokol til Spillepladen

På Figur 51 ses eksempler på anvendelse af de tre første kommandoer i protokolen.

Først ses kommando og retur for Status. Der returneres en tekststreng med status for de forskellige index på pladen i rækkefølgen lav-høj indexering. Der skelnes *ikke* mellem kryds og bolle.

Hernæst vises tilsvarende for Monitor1. Der returneres en tekstreng som indeholder index for ændringen (1) og den nye tilstand på dette index(1). Kommandoen bliver anvendt til Finite-mode og de seks første træk af Infinite-mode.

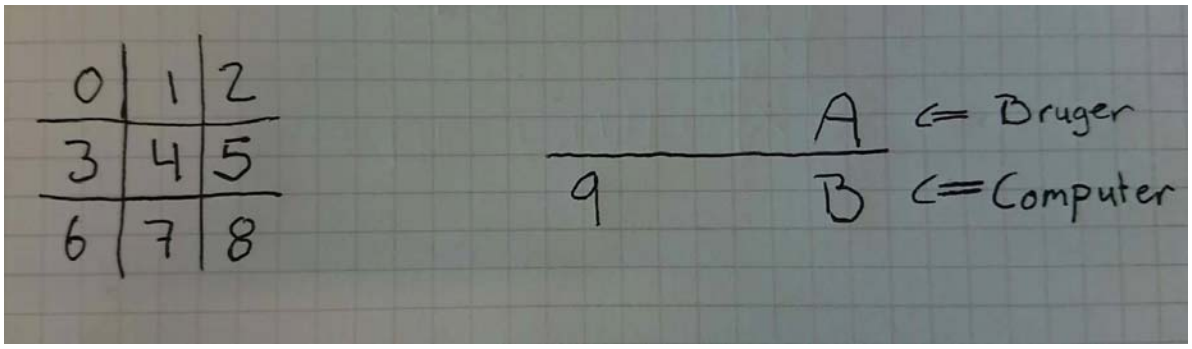
For Monitor2 returneres index for første ændring, ny status på dette index, index for anden ændring og ny status på dette index. Kommandoen benyttes i infinite mode når antal træk er større end 6.

Arm:

Wave
Kommando-byte: 'W'
Data-bytes: ingen
Retur: 'D' eller 'E'
Beskrivelse: Kommandoen bruges under initiering. Vinker med Armen (hardcodede koordinater på PSoC'en). Returnerer 'D' for done, hvis alt går godt, og 'E' for error, hvis noget går galt.

makeMove
Kommando-byte: 'M'
Data-bytes: [fra,til]
Retur: 'D' eller 'E'
Beskrivelse: Kommandoen benyttes for at få Armen til at flytte en brik fra en position til en anden. Positionerne er nærmere specificeret på figurX. Armen returnerer et 'D' for done, hvis alt går godt, og 'E' for error, hvis noget går galt.

Eksempel:



Figur 52: Seriel protokol til Armen

I Figur 52 kan der ses eksempler på anvendelse af de to signaler.

Signalet W[] får blot Armen til at vinke. Dette bliver anvendt i UC1, for at vise at der er skabt kommunikation til Armen.

M[fra, til] skal have to parametre. Første parameter - altså *fra* - er hvor en brik skal hentes fra, og næste parameter - altså *til* - er pladsen hvortil denne brik skal rykkes. Hvis en ny brik skal placeres på plads 0, skrives M[9,0]. Hvis en af brugerens brikker skal sættes på plads, skrives M[plads, A]. Strengen der sendes formateres med c-funktionen snprintf(). Her ses et eksempel på formatering af strengen:

len = snprintf(buf, sizeof(buf), "+M %s %s-", pos1, pos2)

Der laves mellemrum mellem de to positioner, så positionerne kan læses af APSoC'en med sscanf().

4.2 Wi-Fi protokol

Den trådløse Wi-Fi protokol, er protokollen der skal skabe forbindelse mellem Master og GUI.

GUI:

GUI sender et array afsted til Master på tre pladser. Disse tre pladser består alle af en integer-værdi. Som set på Tabel 10 indikerer første/Most Significant bit hvilken mode der spilles i. Næste ciffer viser sværhedsgraden og sidste/Least Significant bit indikerer om det er spilleren eller armen der starter spillet.

Array:	**Første**	**Anden**	**Tredje**
Betydning:	*Spillemode*	*Sværhedsgrad*	*Spilstarter*
Værdi:	1: Finite Mode 2: Infinity Mode 3: Test af Finite Mode 4: Test af Infinity Mode	1: Easy 2: Medium 3: Hard	1: PC starter 2: Spiller starter

Tabel 10: Wi-FI protokol for GUI

Master:

Master sender et array afsted til GUI på to til ubestemt pladser. Disse pladser består hver især af en integer-værdi, og betydningen af disse er beskrevet i Tabel 11. MSB indikerer hvorvidt spilleren har vundet, tabt eller om der står uafgjort. Hvis der er opstået en fejl vil denne værdi blive sendt som et tre-tal. De næste cifre der kommer viser scoren som spilleren har opnået.

Array:	**Første**	**Næste**
Betydning:	*Vinder-status*	*Score*
Værdi:	0: Spiller har tabt 1: Spiller har vundet 2: Uafgjort 3: **Der er opstået en kritisk situation**	Opnået score for spilleren

Tabel 11: Wi-Fi protokol for Master