Aarhus Universitet

3. Semesterprojekt

Gruppe 7

Arkitektur

Beerpong Table

Gruppemedlemmer:
Aaron Becerril Sanchez
(AU592162)
Nikolaj Holm Gylling
(AU592243)
Mathias Magnild Hansen
(AU520773)
Edward Hestnes Brunton
(AU576633)
Marcus Gasberg
(AU587414)
Martin Gildberg Jespersen
(AU593618)
Tristan Moeller
(AU569046)

Vejleder:
Martin Ansbjerg Kjær
Civ. Ing., Ph.D., Adjunkt

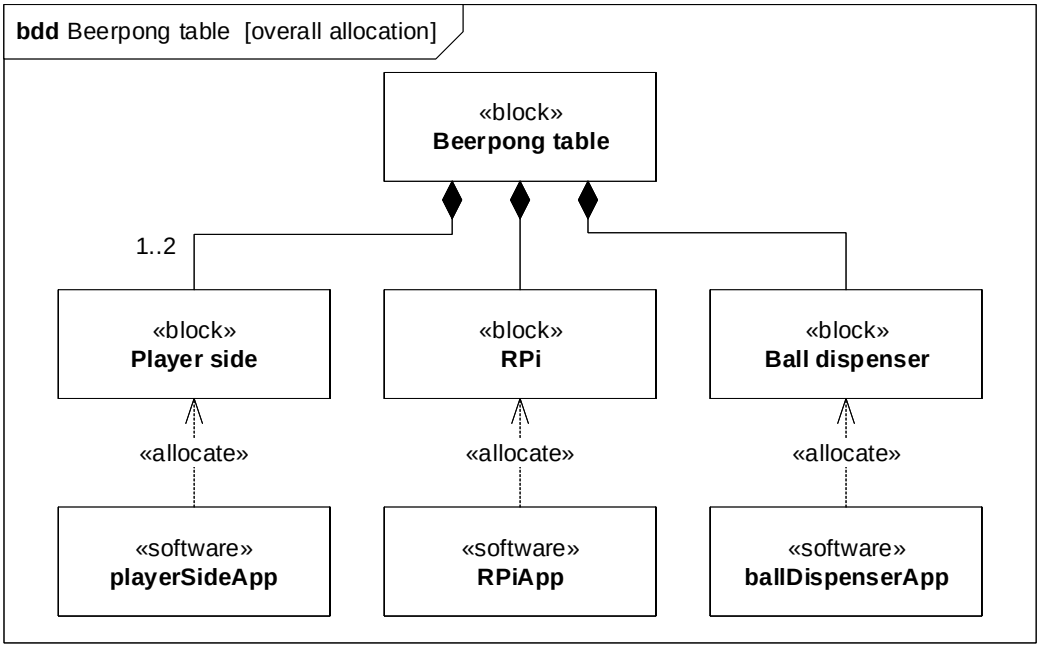
December 2018

Indhold

I det følgende gives en beskrivelse af arkitekturen for Beerpong Table. Formålet med arkitekturen er at beskrive, hvordan systemet er opbygget, og hvordan dets forskellige elementer fungerer. For at give en grundig beskrivelse af systemets funktionalitet og sammensætning, er arkitekturen opdelt i sekvensdiagrammer og henholdsvis system-, hardware- og softwarearkitektur.

1 Systemarkitektur

Indledningsvis er kombinationen af hardware og software vist i et deployment view i figur 1.1. Det viser hvordan softwareapplikationer er allokeret på de tre dele af systemet, som indeholder CPU'er. Det er blokkene Player side, RPi og Ball dispenser, som er beskrevet nærmere i den tilhørende blokbeskrivelse.



Figur 1.1: Overordnet blok definitionsdiagram for systemet med software allokeringer. Indeholder kun blokke med CPU

1.1 Blokbeskrivelse

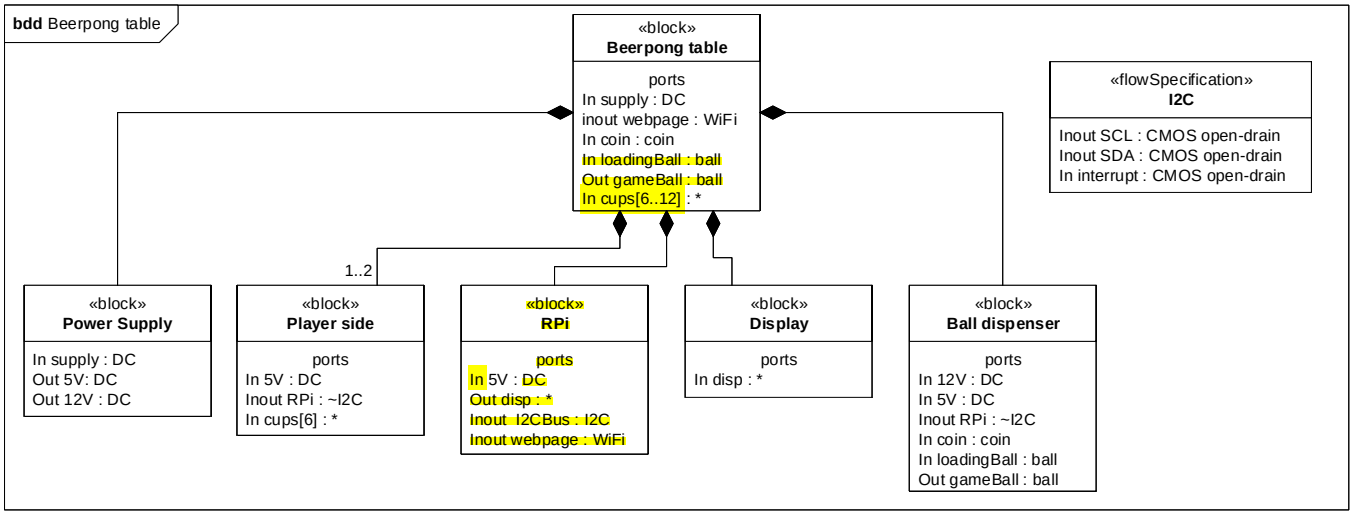
Blok	Beskrivelse
Player side	Skal sørge for at håndtere alt med hensyn til kopperne på den ene ende af bordet. Den skal derfor registrere når en kop fjernes. Og den skal også styre lyset under hver kop. (Evt. skal den også registrere når en bold rammer i en kop) CPU'en i Player side er en PSoC.
RPi	Har til ansvar at styre spillets gang. Den får informationer om de forskellige sensorer fra de andre CPU'er i systemet. Derudover får den også information fra en hjemmeside som den hoster. Den skal på baggrund af disse informationer styre de forskellige aktuatorer vha. de andre CPU'er. Den skal også styre hvad der vises på en skærm. CPU'en i RPi er en Raspberry pi zero W
Ball dispenser	Skal modtage mønter fra brugeren og fortælle RPi om dette. Den skal levere en bordtennisbold når den får besked om det fra RPi. Når den er løbet tør for bolde, skal den informere RPi om dette. Den skal også have lys til at informere bruger og servicemedarbejder om det. CPU'en i Ball dispenser er en PSoC.

2 Hardwarearkitektur

I dette afsnit beskrives hardwarearkitekturen for Beerpong Table. Systemets struktur er illustreret ved hjælp af blokdefinitions diagrammer (BDD). De viser hvordan systemet er sammensat, herunder hvilke dele det består af, og antallet af dele. Et overordnet BDD for systemet er vist i figur 2.1. Blokkene og deres funktion er nærmere specificeret i en blokbeskrivelse. Derudover er der lavet et overordnet internt blokdiagram (IBD), som viser bordets grænseflader, og hvordan blokkene er internt forbundet med hinanden. Særligt blokkene Player side og Ball dispenser er essentielle for spillets afvikling. Der er udarbejdet separate diagrammer for disse - Player side er vist i afsnit 2.2, mens Ball dispenser kan ses i afsnit 2.3.

2.1 Overordnet

2.1.1 Blok definitionsdiagram - Overordnet



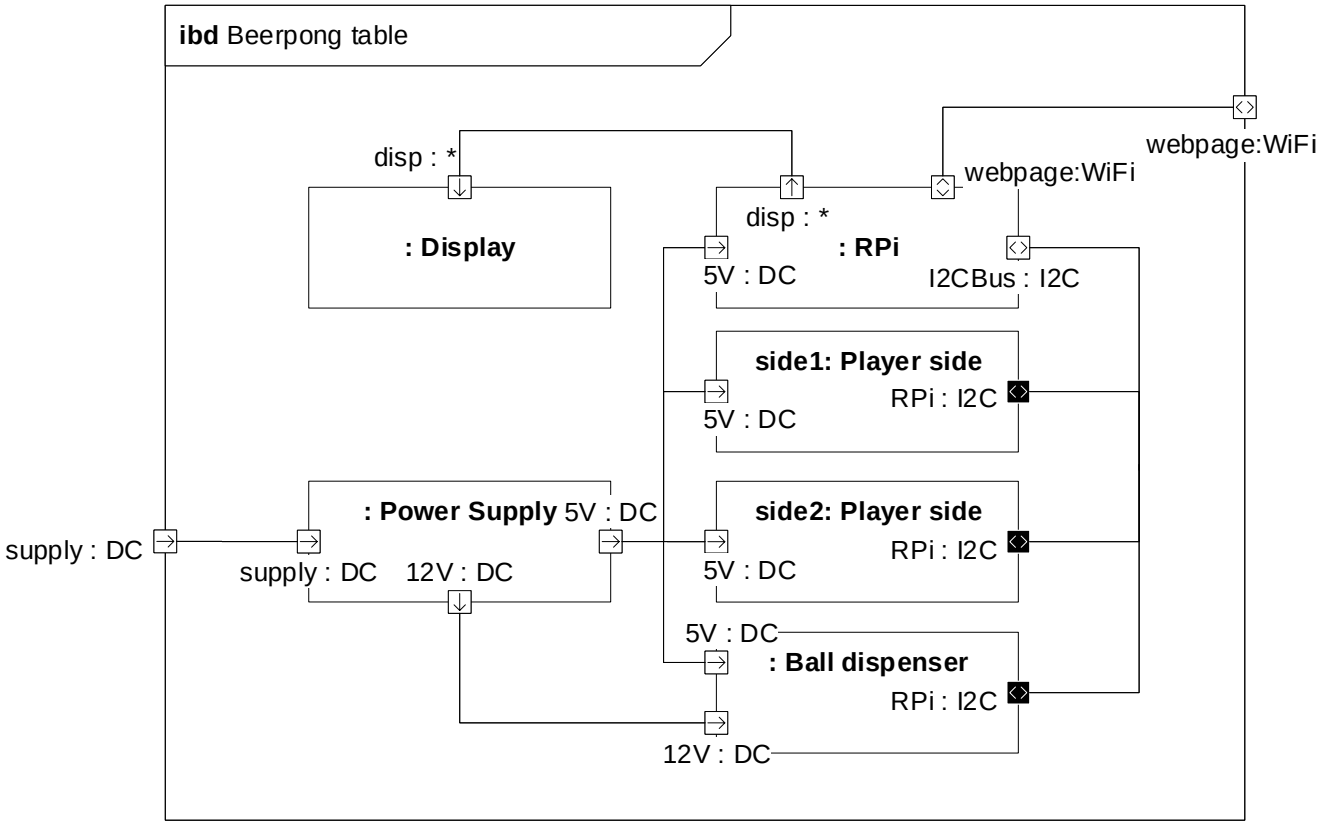
Figur 2.1: Overordnet blok definitionsdiagram for systemet.

2.1.2 Blokbeskrivelse - Overordnet

Blok	Beskrivelse
Power Supply	Hele systemet forsynes med en forsyningsspænding. Denne spænding skal laves om til laves om til forskellige spændinger som de forskellige blokke skal bruge. Dette er Power Supply's ansvar.
Player side	Se afsnit *Systemarkitektur*
RPi	Se afsnit *Systemarkitektur*
Display	Er placeret i midten af bordet og skal vise information til brugerne. Er forbundet til RPi, og det er vises er styret af den.
Ball dispenser	Se afsnit *Systemarkitektur*

2.1.3 Intern blokdiagram - Overordnet

På figur 2.2 ses et ibd for det overordnede system, hvor signaler er vist.

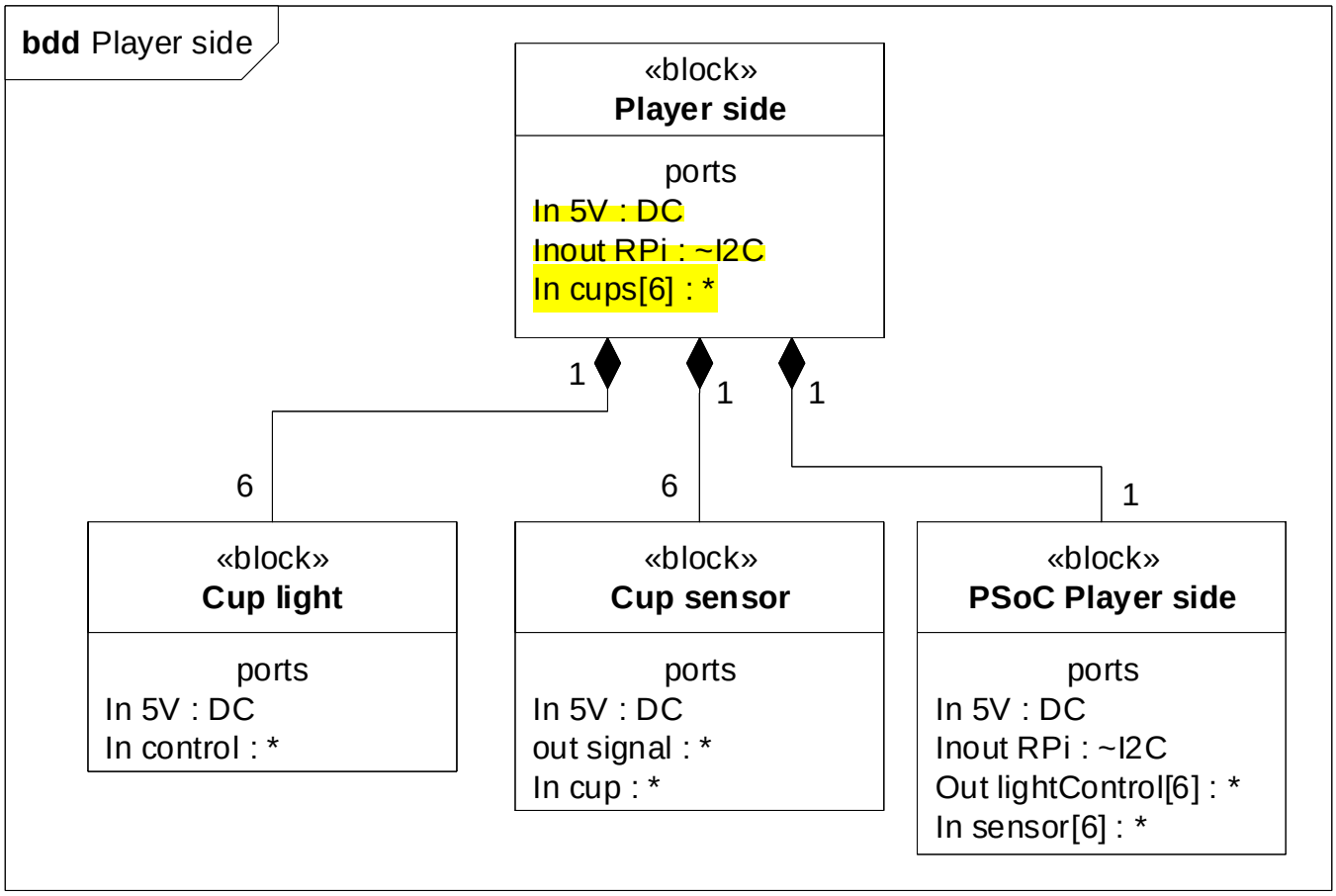


Figur 2.2: Overordnet intern blokdiagram for systemet.

2.2 Player side

2.2.1 Blok definitionsdiagram - Player side

På figur 1.3 ses BDD for player side. Player side findes findes på begge sider af beer pong bordet. Det vil sige at der er en PSoC for hver side af beer pong bordet, sammen med 6 kop sensorer og 6 kopholdere med lys.



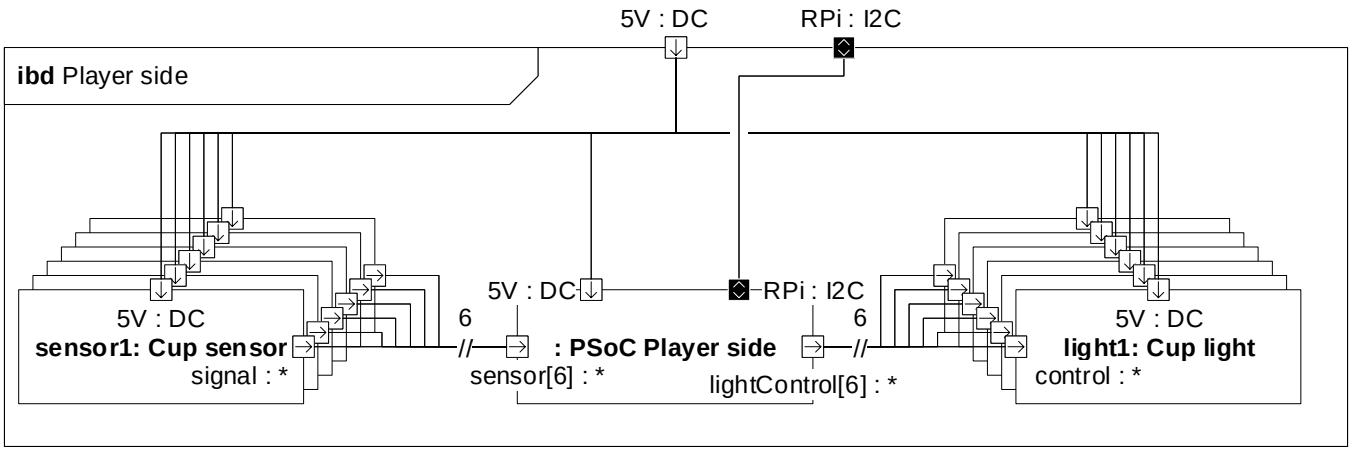
Figur 2.3: Blok definitionsdiagram for Player side.

2.2.2 Blokbeskrivelse - Player side

Blok	Beskrivelse
Cup light	Skal lyse under en kop. Styres af PSoC Player side, og ændres alt afhængig af om kop er placeret på den pågældende kopholder og spillets stadie.
Cup sensor	Skal detektere om der er en kop eller ej på et bestem område af bordet (hvor der er plads til én kop)
PSoC Player side	Skal håndtere sensor input fra 6 Cup sensor's og styre lys på 6 Cup light's. Dette skal ske på bagrund af hvad den får at vide fra RPi. Den skal også sende sensor data til RPi

2.2.3 Intern blokdiagram - Player side

Nedenfor på figur 1.4 ses et ibd for player side, hvor man ser PSoC'en i midten, som checker for inputs fra kop sensorerne. Sensorerne ses på venstre side i ibd'et. Alt afhængig af inputs fra sensorerne vil PSoC'en opdaterer Kop LED'erne. Hver sensor styrer indirekte LED'ens status.

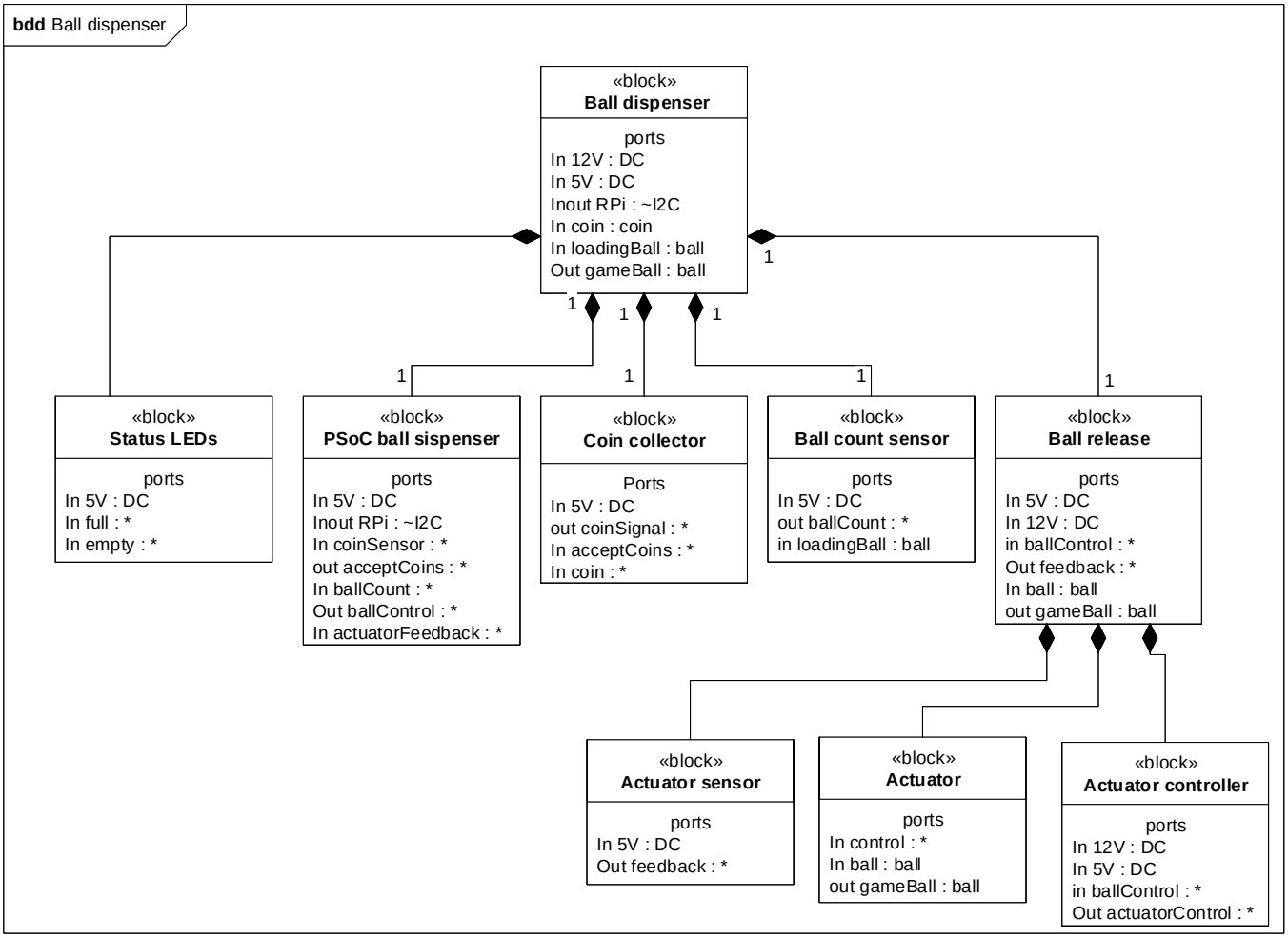


Figur 2.4: Intern blokdiagram for Player side.

2.3 Ball dispenser

2.3.1 Blok definitionsdiagram - Ball dispenser

På figur 1.5 ses et BDD for ball dispenser. Der er kun en ball dispenser i hele systemet. Ball dispenser består af en coin collecter, hvor 5 krone mønten man skal starte spillet med skal indsættes, hvorefter to bolde vil dispenseres. Dette gøres gennem Ball release, der består af en motor(aktuator), som står for dispenseringen. En ting der er vigtig for systemet er at vide, hvornår der er nok bolde i ball dispenseren. Dette gøres ved hjælp af underblokken ball count sensor. Her vil en sensor sørge for at der er mindst to bolde klar til dispensering hele tiden ellers lyser en status LED rødt. Hertil vil en status LED lyse grønt, hvis ball dispenseren er fuld. Samspil mellem alle blokkene styres af PSoC'en.



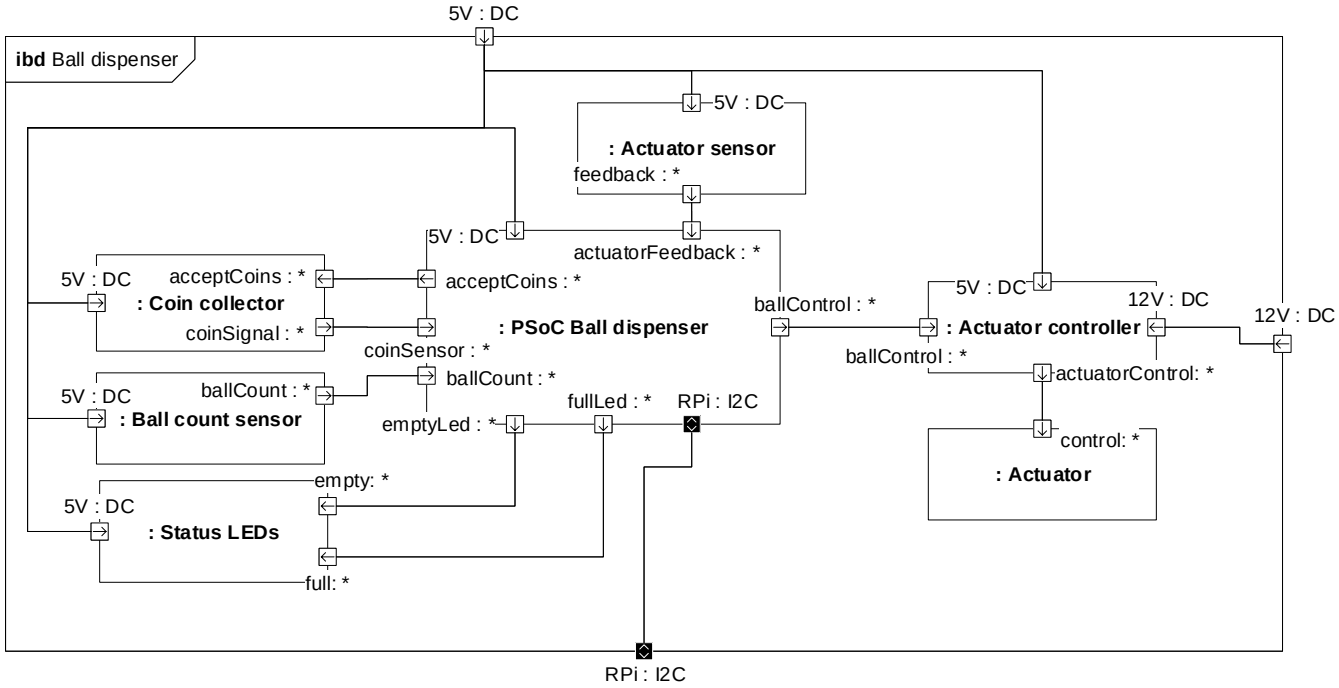
Figur 2.5: Blok definitionsdiagram for Ball dispenser.

2.3.2 Blokbeskrivelse - Ball dispenser

Blok	Beskrivelse
Coin collector	Skal detektere når der indsættes mønter. Den skal detektere 5kr mønter og andre skal returneres til brugeren. Derudover skal 5kr mønten også returneres hvis systemet ikke er i en tilstand hvor det er klar til at modtage mønter. Fx at der er et spil i gang eller hvis der ikke er flere bolde.
Ball count sensor	Skal holde styr på hvor mange bolde der er i dispenseren
Ball release	Skal sørge for at levere en/flere bold(e) til brugeren
Actuator	Skal lave bevægelsen der leverer en bold
Actuator sensor	Skal detektere positionen af Actuator. Dette bruges til at styre Actuator.
Actuator controller	Skal styre Actuator.
PSoC ball dispenser	Skal håndtere sensor input fra Coin collector. Og skal afhængig af information modtaget fra RPi styre om Coin collector skal returnere mønter. Den skal også få Ball release til at levere en bold når den får det at vide af RPi. Derudover skal den også fortælle RPi om hvor mange bolde der er tilbage.
Status LEDs	To LED'er til at informere om bold dispenseren er fuld eller tom. En rød LED lyser når den er tom (under 2 bolde tilbage) og en grøn LED lyser når bolddispenseren er fuld

2.3.3 Intern blokdiagram - Ball dispenser

På figur 1.6 ses et ibd diagram for Ball dispenseren, hvor man her ser, at PSoC'en har kontakt til Raspberry pie, og sørger for samspillet mellem coincollector, Ball count sensor og motoren aktuator. Man kan her se at ball release blokken ikke længere er med fra BDD diagrammet. Derimod er de blokke den bestod af med, som er aktuatoren(den bevægelige del), en actuator controller til styring af den bevægelige motor og til sidst en sensor til at holde styr på motorens(aktuatorens) position.



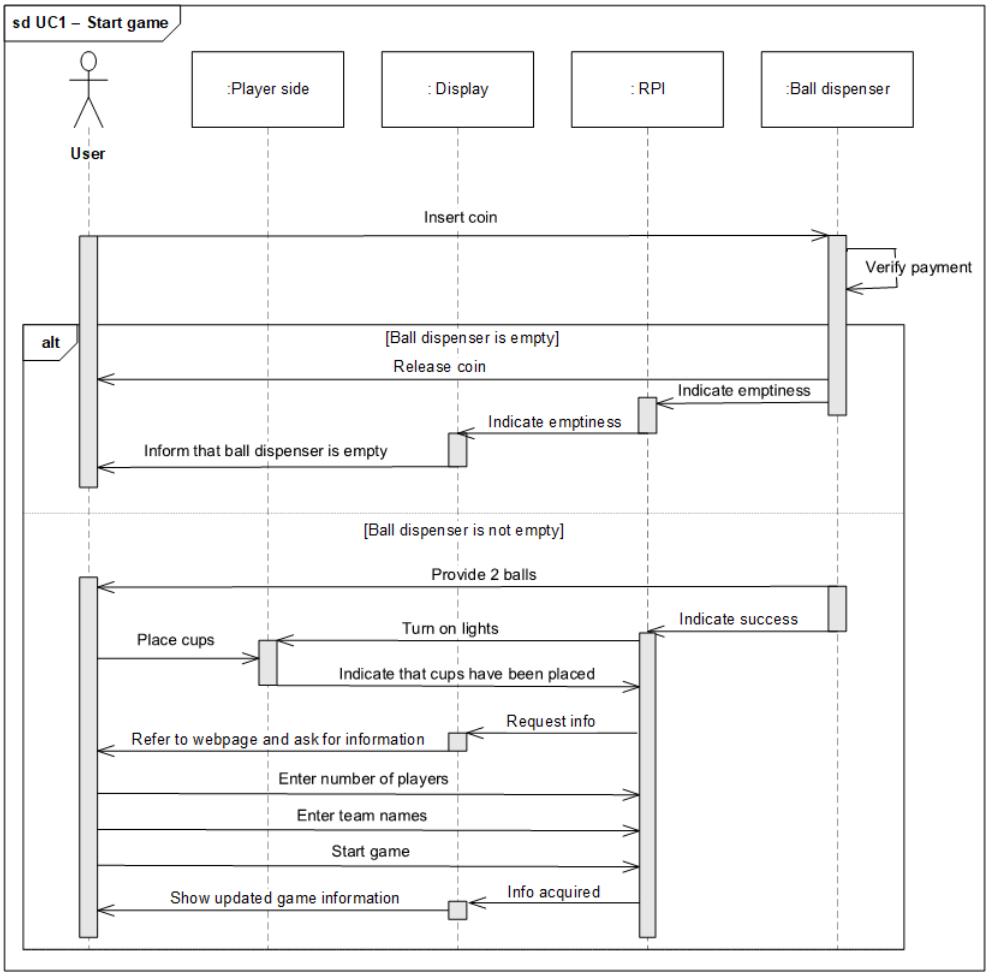
Figur 2.6: Intern blokdiagram for Ball dispenser.

3 Sekvensdiagrammer

I dette afsnit er der vist sekvensdiagrammer (SD) for systemet. De er udarbejdet med udgangspunkt i Use Cases, og de viser, hvordan aktørerne interagerer med systemet, og hvordan de forskellige brugsscenarier forløber. Sekvensdiagrammet for 'UC1 - Start game' er vist i figur 3.1, efterfulgt af 'UC2 - Play turn' i figur 3.2. Endelig er 'UC3 - End game' og 'UC4 - Refill balls' vist i hhv. figur 3.3 og 3.4.

3.1 Sekvensdiagram for UC1

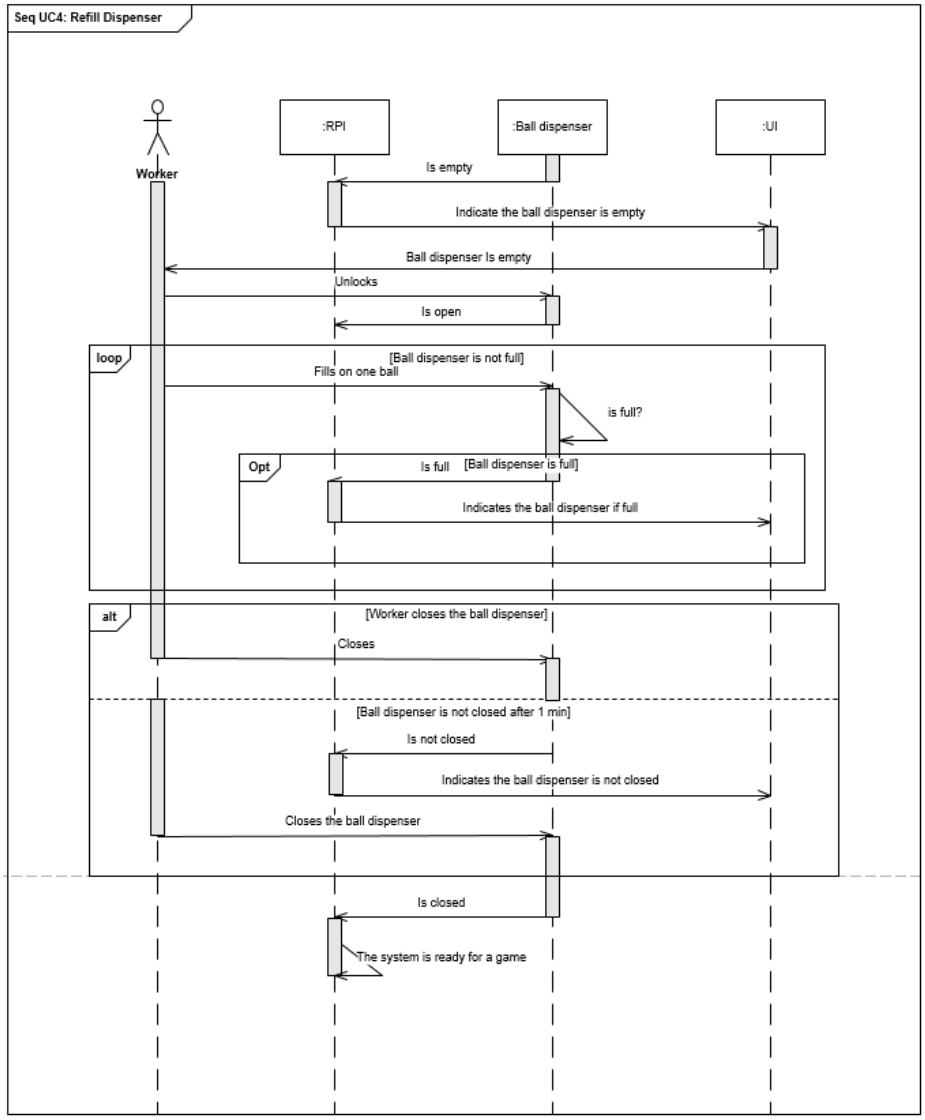
Start game sekvensen beskriver hvordan bruger og system burde sætte spillet op. Sådan så alt er fuldfunktionelt og klar til at køre et helt beer-pong spil igennem.



Figur 3.1: Sekvensdiagram for UC1 - Start spil

3.2 Sekvensdiagram for UC2

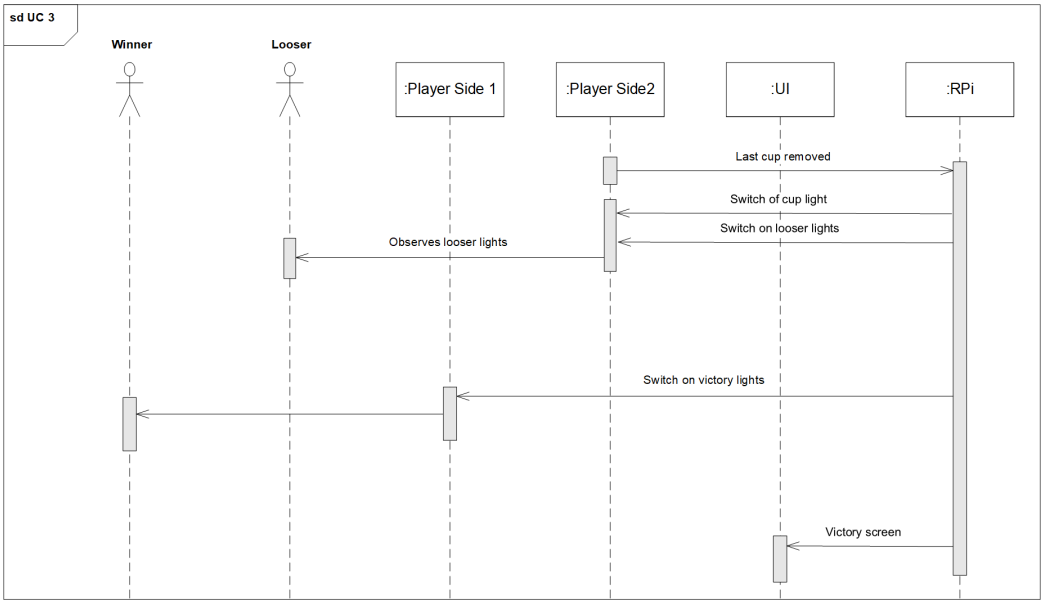
Sekvensen Play turn bekriver spillets gang og overdragelse af ***tur***.



Figur 3.2: Sekvensdiagram for UC2 - Play Turn

3.3 Sekvensdiagram for UC3 End game

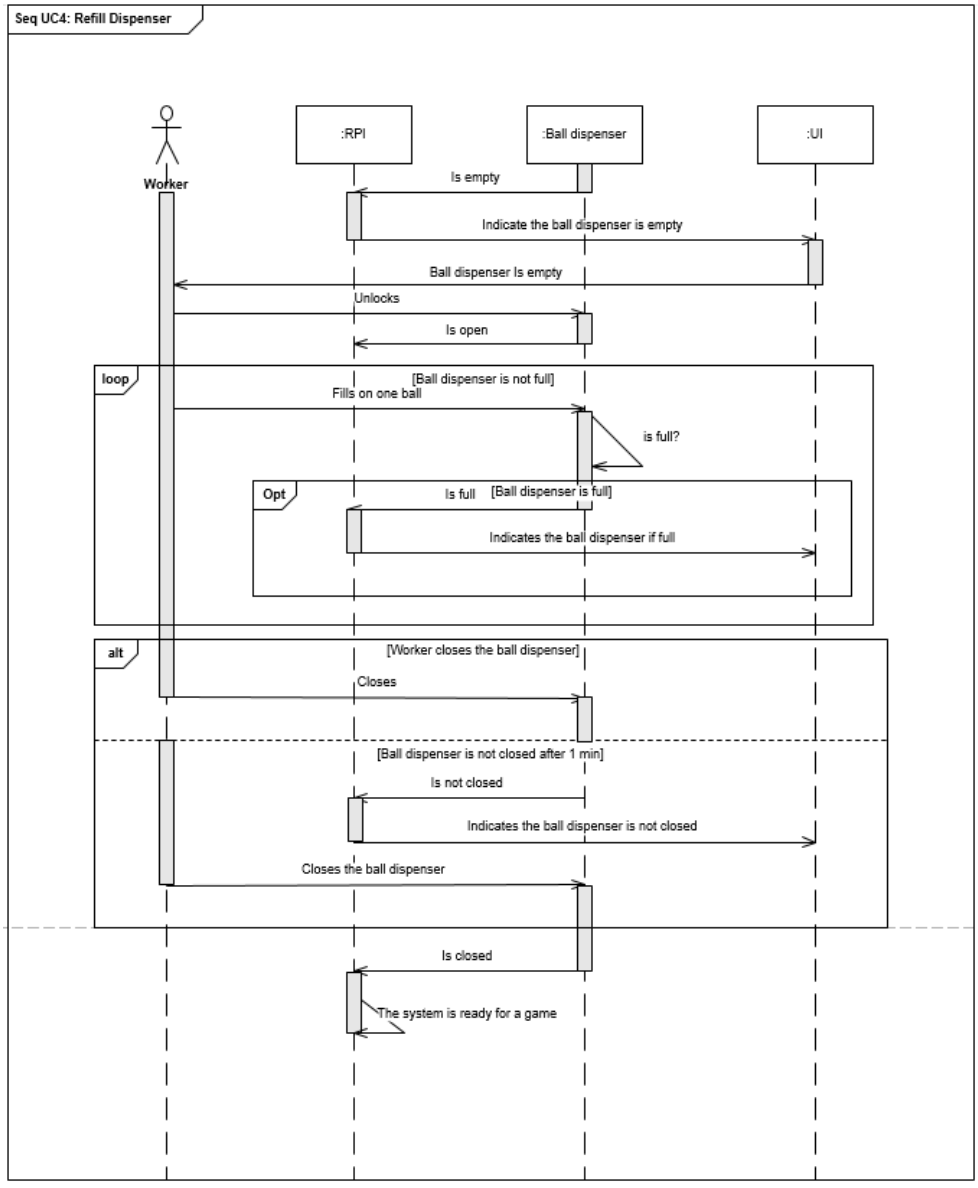
End game sekvens skal læses som, det sekvens der køres når et hold har vundet og skal dermed fejres.



Figur 3.3: Sekvensdiagram for UC3 - End Game

3.4 Sekvensdiagram for UC4 Refill Dispenser

Sekvensen Refill Dispenser står for sig selv som en service funktion og er uafhængig af de andre sekvenser.



Figur 3.4: Sekvensdiagram for use case 4: Påfyld bolde

4 Grænseflader

This section explains the different communication protocols used in the Beer-pong Master 2000.

4.1 Communication between Raspberry Pi and Player Side

The communication between the RPi and the player side is done using the I2C communication protocol. I2C is a two way synchronous communication protocol.
The communication starts when the RPi receives an active low interrupt from one of the PSoCs and keeps polling them all until all the data is read and the interrupt signal returns to high.

Bit 1	**Bit 2 - 8**	**Bit 9**	**Bit 10**	**Bit 11 - 19**	**Bit 20**	**Bit 21 - 29**	**Bit 30**	**Bit 31**
Start bit	Slave Address	R/W	ACK	Data Bits	ACK	Data Bits	NACK	Stop Bit

Slave Address: This is the address of the PSoC we are communicating with. So in our case it will be ether one of the player sides or the ball dispenser.
R/W: This is the bit that determines if the master is going to read or write to the slave.
ACK: This is a response from the slave to confirm that it has received the command.
Data Bits: The data being transferred.
NACK: A signal that the communication is over.

4.1.1 The Protocol

Command	Purpose
EnableCollector	Enables the coin collector
DisableCollector	Disables the coin collector
CoinInserted	A coin is received by the collector
CupStatus	The PSoCs in the playerside sends the status for every cup on its side
DispenseBall	Tells the ball dispenser to dispense one ball
DispenserEmpty	Signals that there are no more balls in the dispenser
DispenserFull	Signals that the balls have been refilled
StartGame	Changes all PSoCs mode to startup mode
GameInProgress	Changes all PSoCs mode to "playgame"mode
SetModeVictory	Changes a PlayerSide to victory mode
SetModeDefeat	Changes a PlayerSide to Loser mode
SetModeIdle	Changes all the PSoCs to idle mode
TeamColour	Transfers the selected team colours to the PSoC

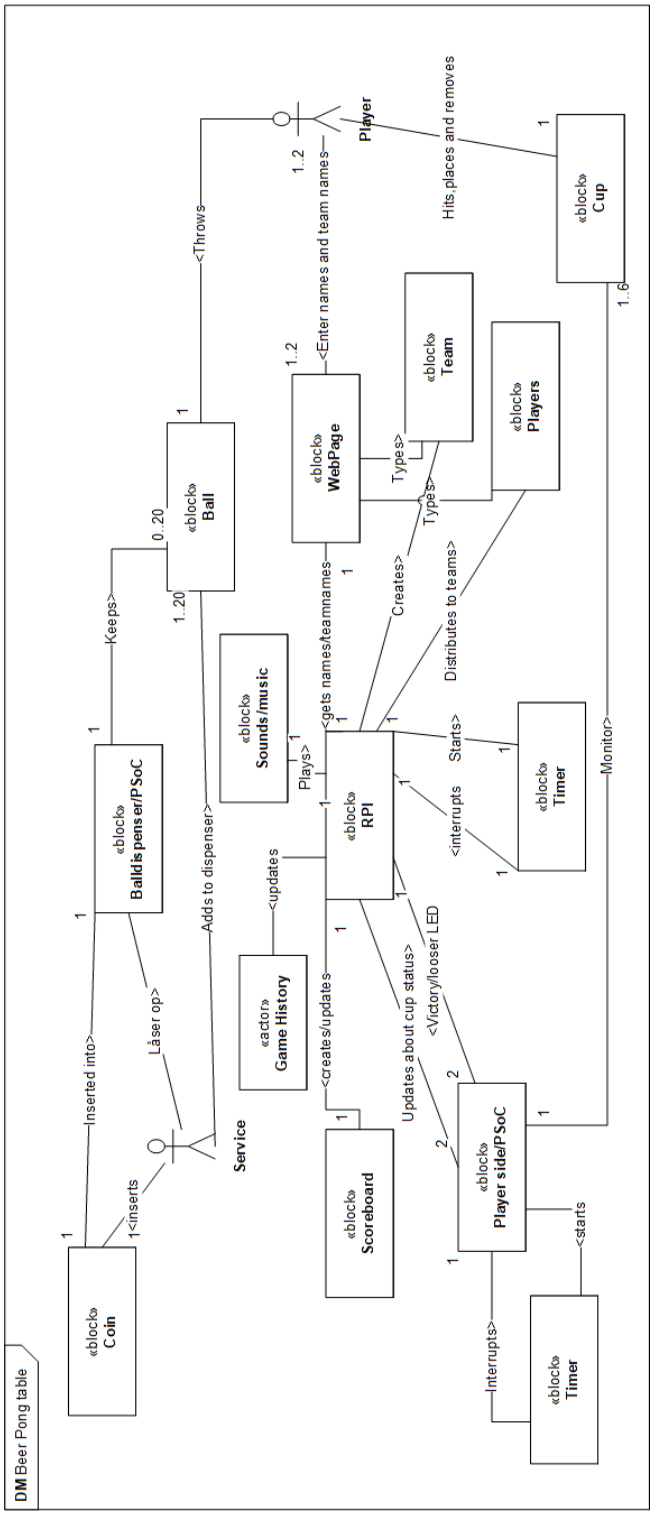
5 Softwarearkitektur

I dette afsnit beskrives systemets softwarearkitektur. I figur 1.1 er allokeringen af applikationer for systemet vist, og det fremgår at vi har de tre applikationer playerSideApp, RpiApp og ballDispenserApp. Der vises her applikationsmodeller for hver af disse, samt en overordnet domænemodel for systemet.

5.1 Domænemodel

I det følgende er systemets Use Cases analyseret, med det formål at lave en model, som viser systemets domæne. Domænemodellen i figur 5.1 giver et overblik over, hvilke elementer der indgår i Beerpong Table's software. Det illustrerer systemets funktion på et mere overordnet, konceptuelt plan og viser hvilke elementer, som senere skal designes og implementeres. Det fremgår af figur 5.1, hvordan servicemedarbejder interagerer med Ball dispenser, mens

bruger primært interagerer med kop og WebPage. Derudover ses det bl.a., at RPi er afgørende for afviklingen af WebPage, herunder Scoreboard og Game History, samt kommunikationen med PSoC i Player side.



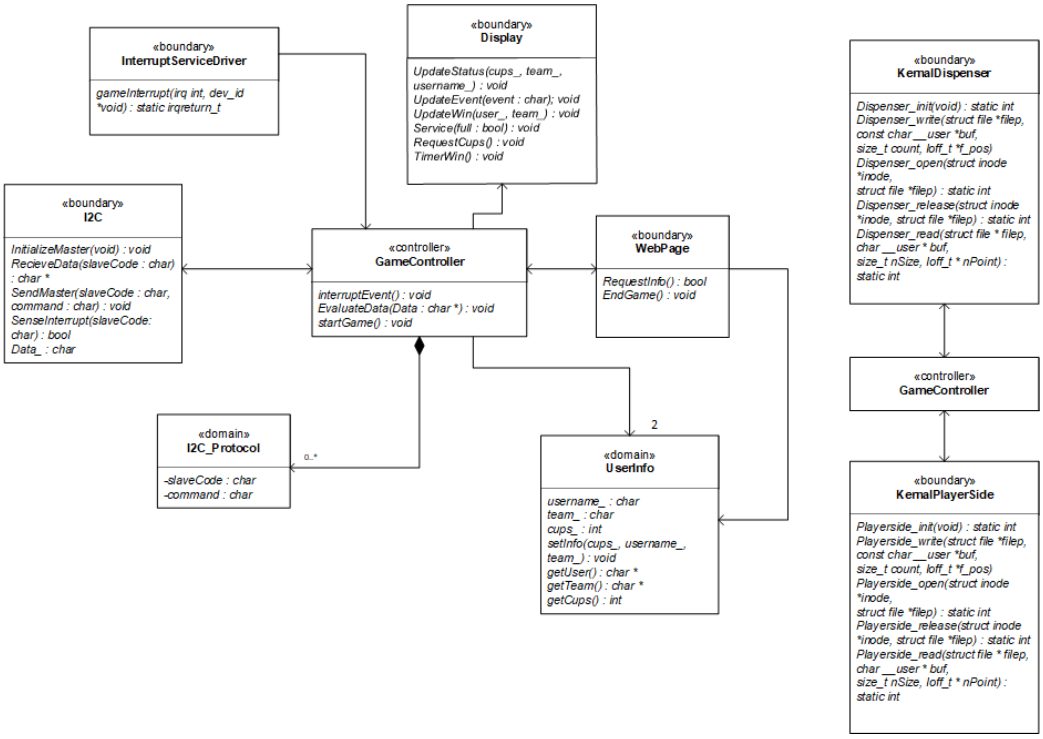
Figur 5.1: Domænemodel for Beerpong Table

5.2 Applikationsmodeller

På baggrund af domænemodellen i figur 5.1, allokeringsdiagrammet i figur 1.1 og Use Cases er der udarbejdes applikationsmodeller for hver software applikation. Det har resulteret i tre applikationsmodeller, hhv. playerSideApp, RPiApp og ballDispenserApp. Modellerne udgør grundlaget for det videre designarbejde og består som regel af et klassediagram, sekvensdiagram(mer) og evt. et state machine diagram.

5.2.1 RPiApp

Dette er første udkast til applikationsmodellen for RPi - Vi arbejder stadig med at finde en løsning til at vise hvordan man skelner mellem user space og kernal space i applikationsmodellen.
En anden ting er, at grænsefladerne og protokollerne er ikke helt opdateret og "slavecodes"er midlertidig navngivet.
SlaveCode:
0xBD = BallDispenser
0XPS = Playerside (Ikke angivet en for hver playerside endnu) For andre kommandoer henvises til grænseflade afsnittet tabel **??**



Figur 5.2: Klassediagram for RPi

Controller: GameController
GameController er den centrale klasse for alle use cases. Den sørger for at sende og modtage data gennem klassen I2C. Alt modtaget data bliver evalueret i GameController og sendt videre i overensstemmelse med protokollen og use case flow. Klassen varetager alt logikken og de flest beslutninger i systemet.

Boundary: Display
Klasse til at opdatere det fysiske display tilknyttet systemet.

Boundary: WebPage
Grænsefladen til brugeren, en hjemmeside som modtager brugernes information og lagre dem i UserInfo.

Domain: UserInfo
Hukommelse om brugerne, hold og antal kopper tilbage. Denne domæne klasse findes ikke i domænemodellen, men er en sammensætning af domæneklasserne players, teams og scoreboard. Der er to af denne klasse, da hvert hold har informationer, og vurderingen om hvem har vundet tages ud fra dette.

Boundary: I2C_Protocol
Information om slavernes adresse og kommandoernes betydning og funktion.

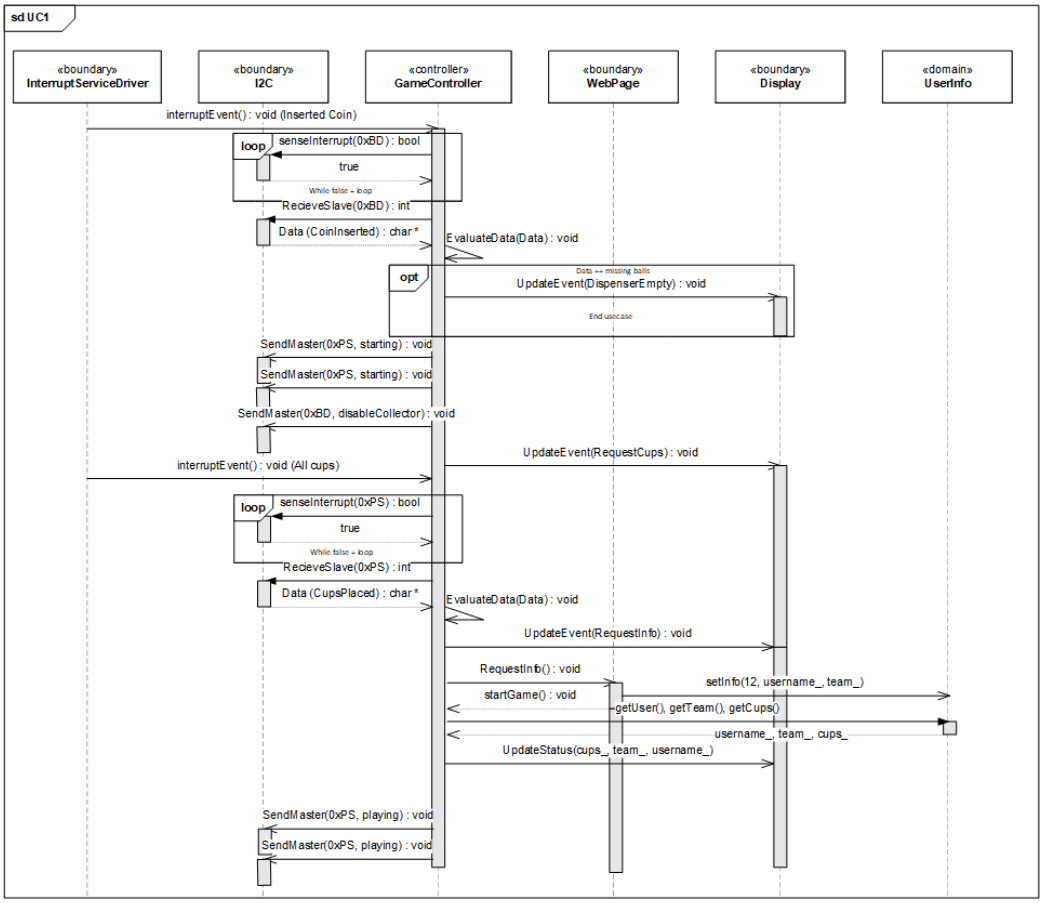
Boundary: I2C
Modtager og sender data mellem de forskellige enheder. Master er RPi og slaverne: Playerside(1-2) og Coin dispenser. For at den kan læse værdien sendt gennem I2C bruger GameController de to drivere "KernalPlayerSide"og "KernalDispenser".

Boundary: InterruptServiceDriver
InterruptServiceDriver er ikke en klasse, men derimod en driver implementeret i kernalspace. Dets ansvar er at "vække" GameController i tilfælde af interrupts fra en slave.

Boundary: KernalPlayerSide
Driver til at læse/skrive til Playerside via I2C protokollen. Enten sendes information fra userspace til kernalspace, som så sendes videre til Playerside - ellers læses data fra Playerside i kernalspace og sendes videre til userspace.

Boundary: KernalDispenser
Driver til at læse/skrive til BoldDispenser via I2C protokollen. Enten sendes information fra userspace til kernalspace, som så sendes videre til BallDispenser - ellers læses data fra BoldDispenser i kernalspace og sendes videre til userspace.

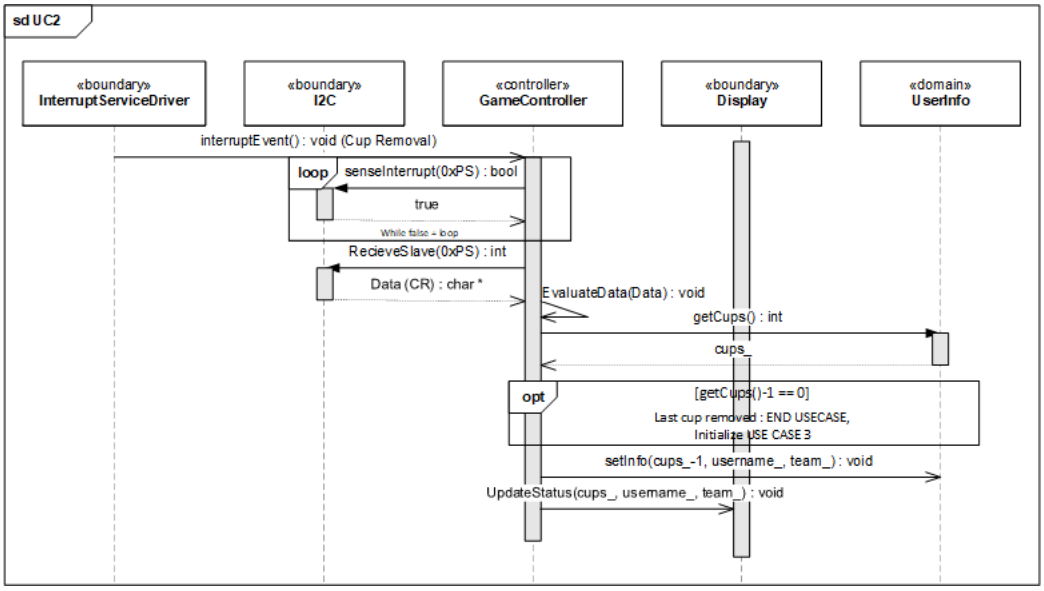
Sekvensdiagram for UC1: GameController
GameControlleren er logikken i systemet og tager alle de store beslutninger. Dens hovedfunktioner er at modtage data fra de enkelte delsystemer, som boldDispenser og Playerside, og agerer i forhold til dette. Hvis et af disse delsystemer ønsker at kommunikere med GameControlleren (RPi) sendes der et ekstern interrupt. GameController poller derefter delsystemerne: den spørger hvem som har sendt interruptet. Når den har fundet ud af dette, så spørger den om den data den ønsker at sende, og opdaterer de andre systemer i forhold til den data.
GameController sender også hvilken stadie Playerside og BoldDispenser skal være i, fx når spillet startes skal de være i stadiet "Starting".



Figur 5.3: Sekvensdiagram for UC1

Sekvensdiagram for UC2: GameController

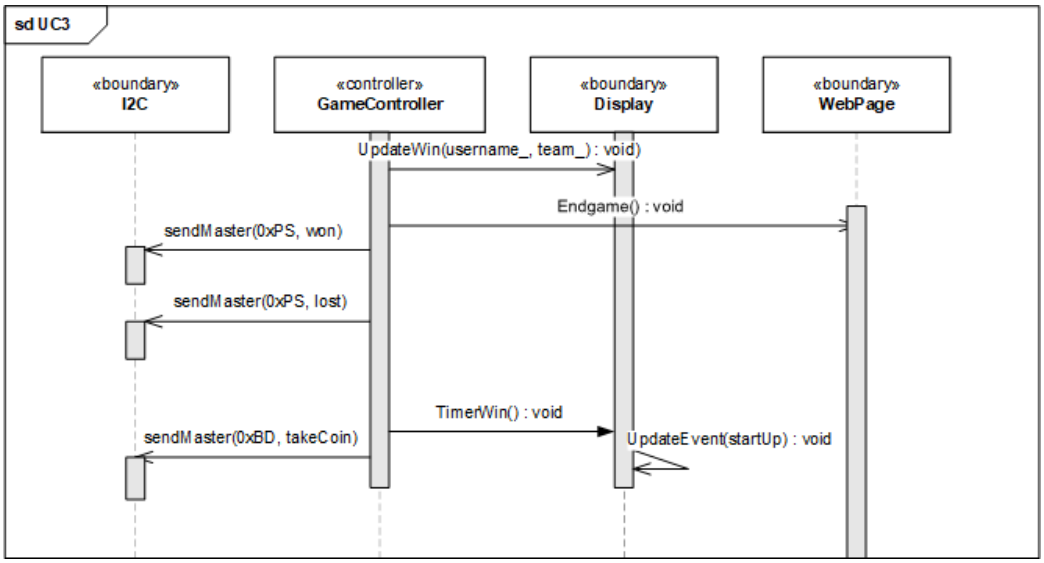
Spillet er startet og GameController har til opgave at hele tiden holde systemet opdateret i forhold til brugerens interaktioner. Hver gang brugeren fjerner en kop modtager RPi'en et interrupt (samme princip som i sekvensdigram for UC1). GameController opdaterer herefter displayet, således det som vises stemmer overens med det som sker på virkeligheden.



Figur 5.4: Sekvensdiagram for UC2

Sekvensdiagram for UC3: GameController

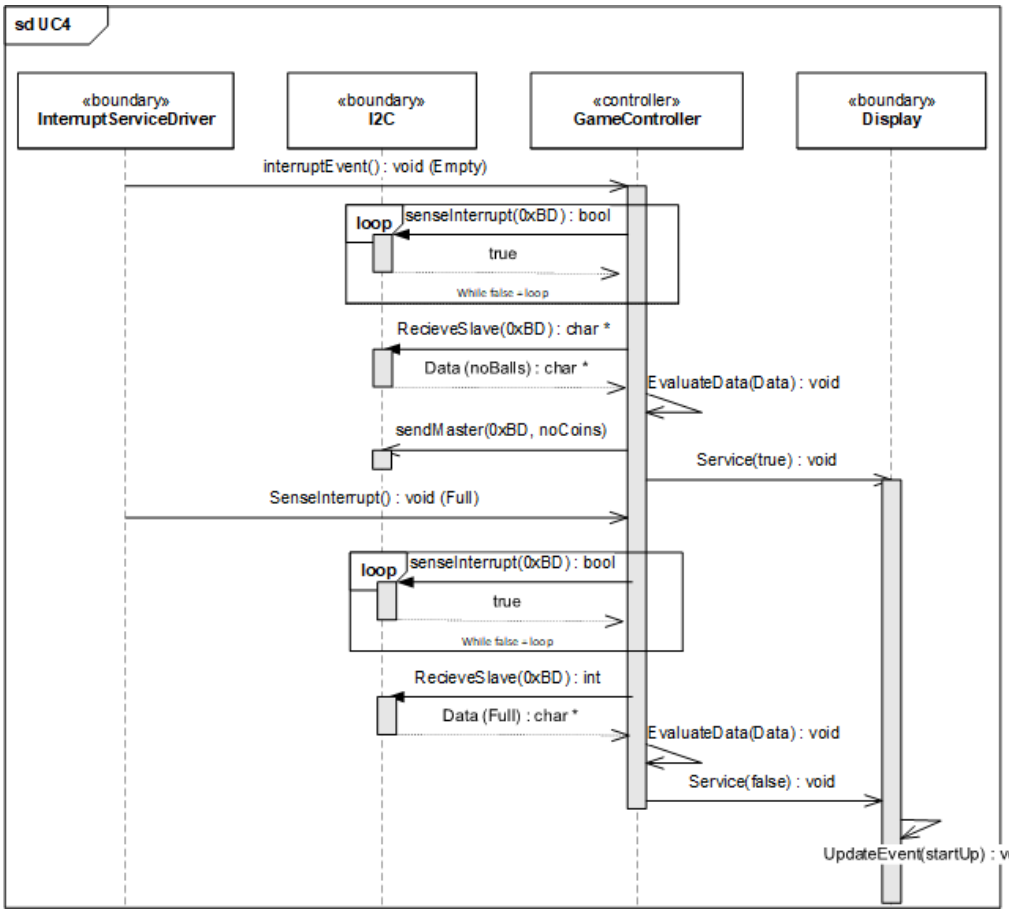
GameControlleren har selv kontrolleret antallet af kopper tilbage på hvert side i forhold til det data, som har været sendt fra de to Playersides. Når den sidste kop er fjernet, sender den henholdsvis et signal til hver Playerside, hvem som har vundet og tabt. Dette vises på displayet i et specifikt tidsinterval, hvor derefter systemet vil gå i dvale og være klar til nye spillere.



Figur 5.5: Sekvensdiagram for UC3

Sekvensdiagram for UC4: GameController

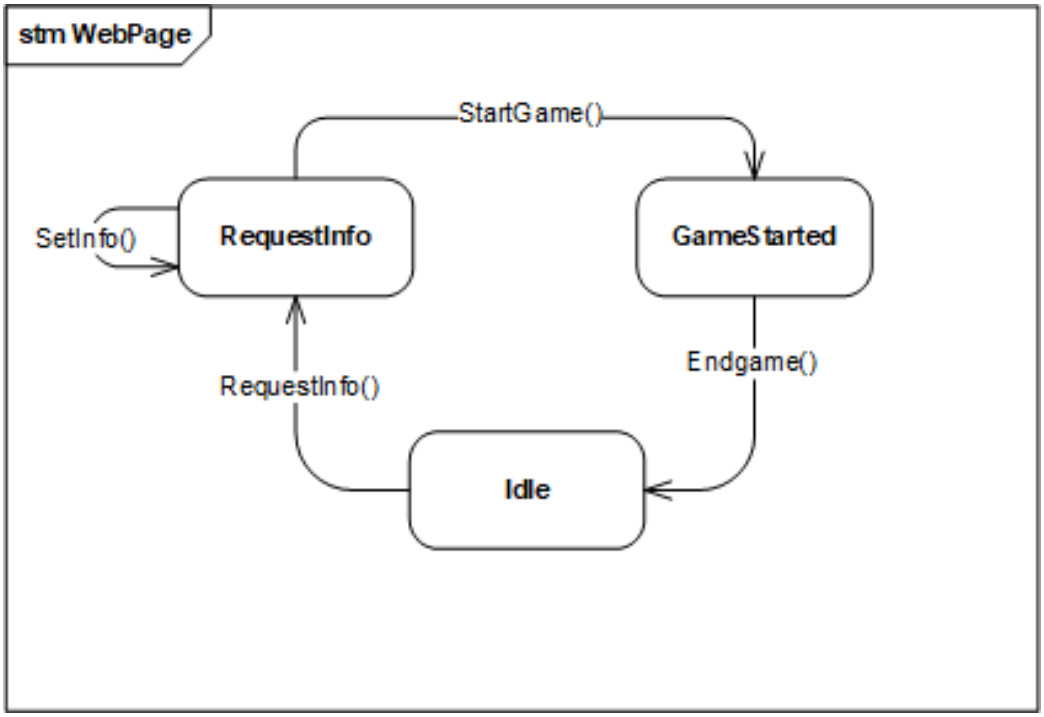
I tilfælde af at boldDispenseren ikke har flere bolde, skal RPi'en udstede en besked til servicemedhjælperen og vente på at den er fyldt op (Eller har nok bolde til et spil).



Figur 5.6: Sekvensdiagram for UC4

Tilstandsmaskine: WebPage

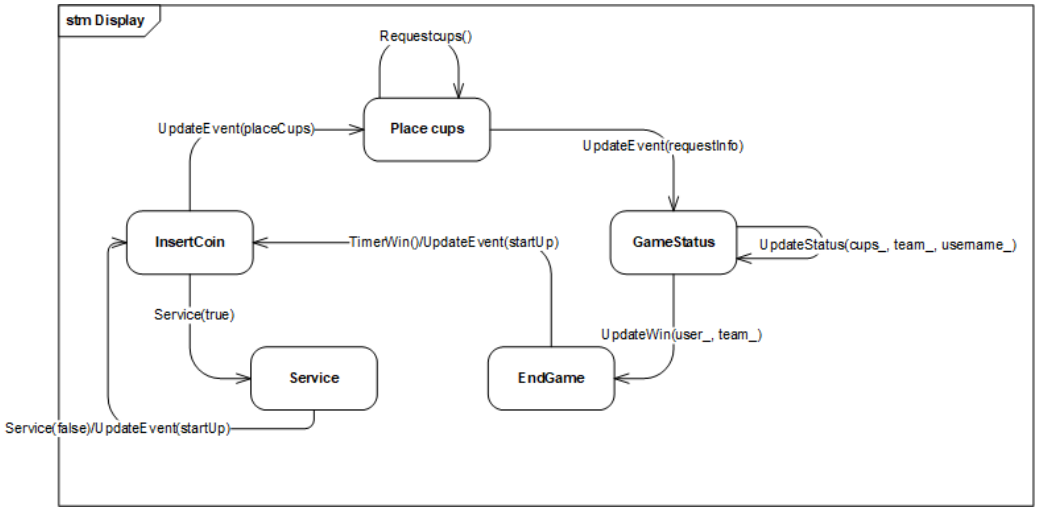
WebPage er brugerens mulighed for at gøre spillet personligt. Her indtastes holdnavne og brugernavne. WebPage er i en dvale tilstand indtil spillet starter. Her bliver brugeren spurgt om denne information gennem GUI'en og givet en IP-addresse via displayet. Når brugeren har trykket på "Start Game"udsteder den en konstant besked indtil spillet er slut.



Figur 5.7: Tilstandsmaskine for WebPage

Tilstandsmaskine: Display

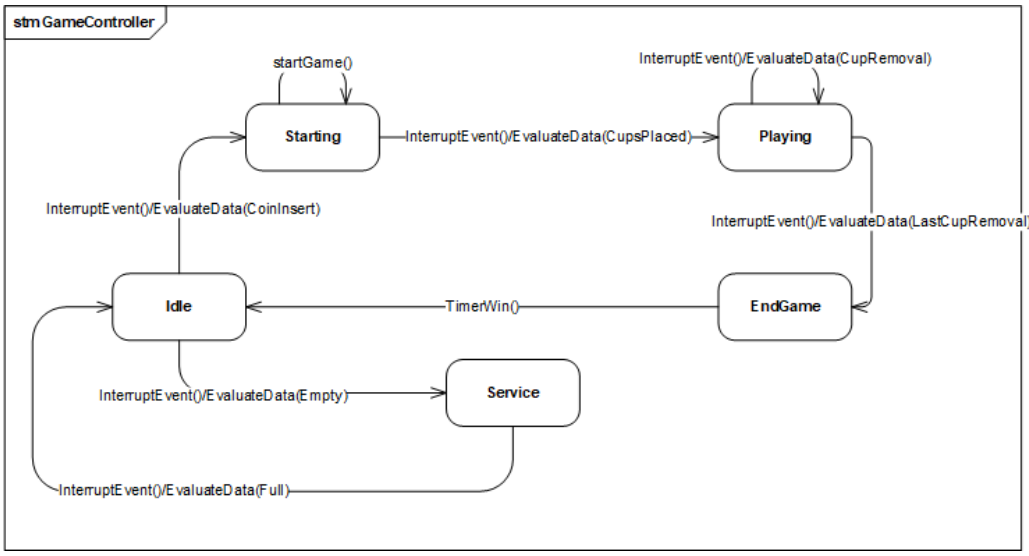
Displayet viser konstant spillets status og fortæller brugerne hvad de skal gøre. Den skifter derved også tilstand i takt med spillets gang.



Figur 5.8: Tilstandsmaskine for Display

Tilstandsmaskine: GameController

Tilstandsmaskinen for GameController afspejler spillets gang. Den skifter konsekvent tilstand for hvert gang den får et interrupt og data fra de 3 delsystemer: Playerside(1 og 2) og BoldDispenser.

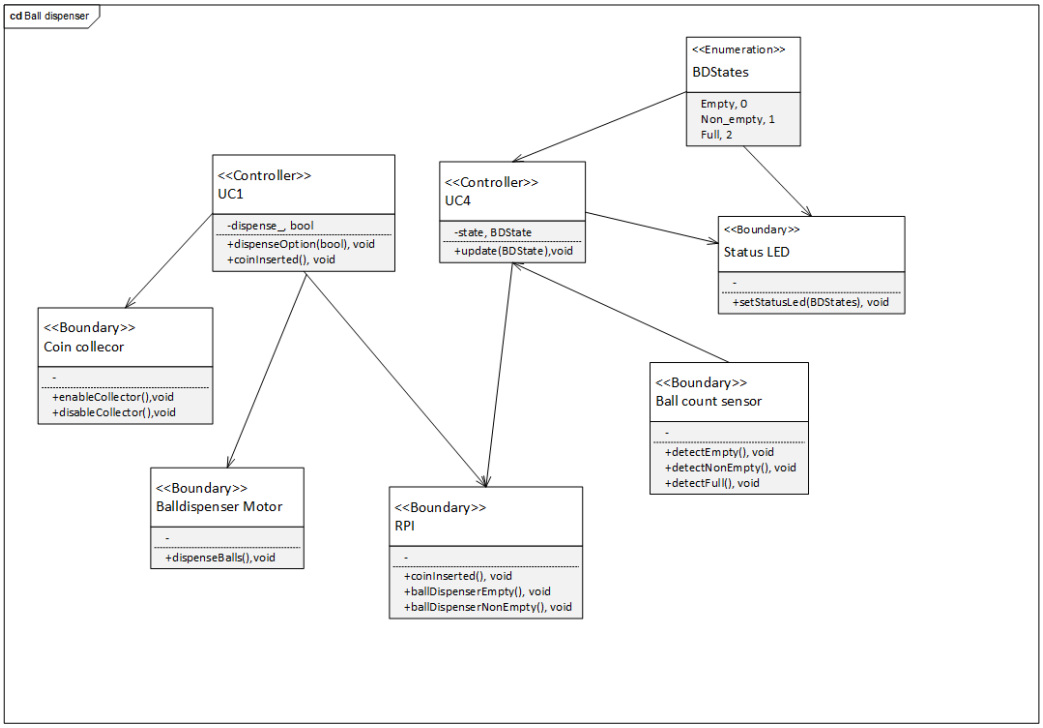


Figur 5.9: Tilstandsmaskine for GameController

5.2.2 Applikationsmodel for Bolddispenser

I dette afsnit præsenteres applikationsmodellen for bolddispenseren, hvor der er lavet et overordnet klassediagram og statemachine for dispenseren, som spiller sammen med et sekvensdiagram for UC1 og UC4, der var de mest relevante i forbindelse med Bolddispenser.

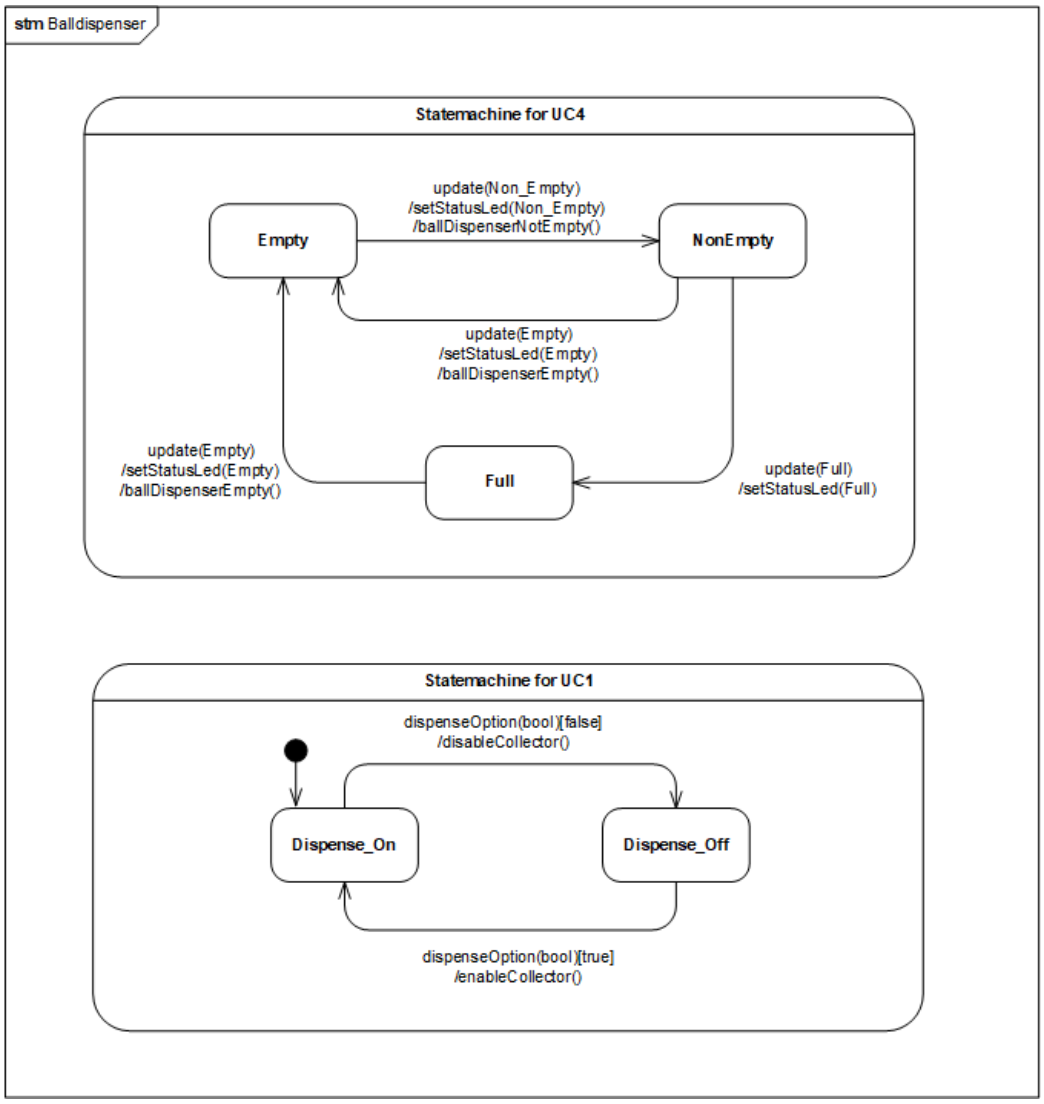
Klassediagram: Bolddispenser
Startes der med klassediagrammet for det overordnede overblik over relationer og metoder, så ses dette i figur 5.10.



Figur 5.10: Class Diagram for bolddispenser

Tilstandsmaskine: Bolddispenser

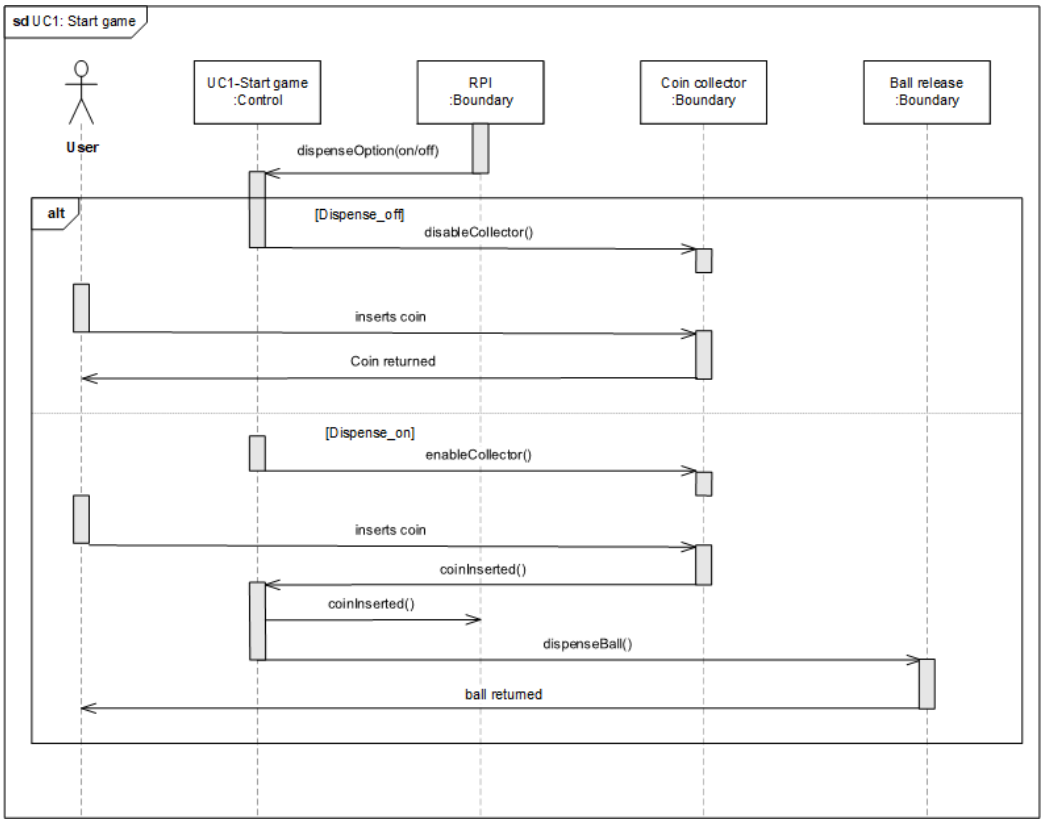
I bolddispenseren kan der deles op i to forskellige typer tilstande. En tilstand om bolddispenserens kapacitet af bolde og en tilstand om bolddispenseren må dispensere bolde. Hvordan bolddispenseren bevæger sig imellem disse tilstande, og hvad der sker kan ses i figur 5.11.



Figur 5.11: Tilstandsmaskine for bolddispenser

Sekvensdiagram UC1: Bolddispenser

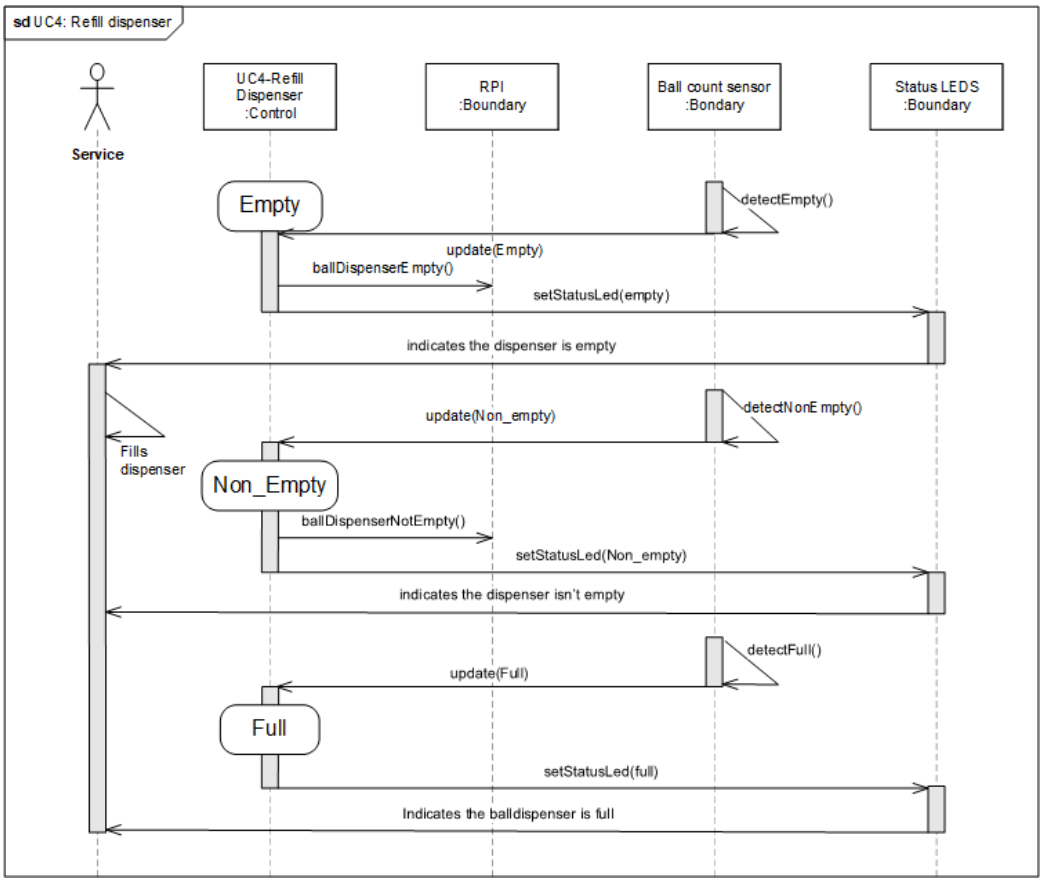
Når et spil skal startes, er det i pricippet bolddispenserens ansvar at starte spillet i det, den skal registrere indsættelsen af en mønt og sende et signal om dette til RPI'en. Forløbet i dette er beskrevet i figur 5.12.



Figur 5.12: Sekvensdiagram for UC1 for bolddispenser

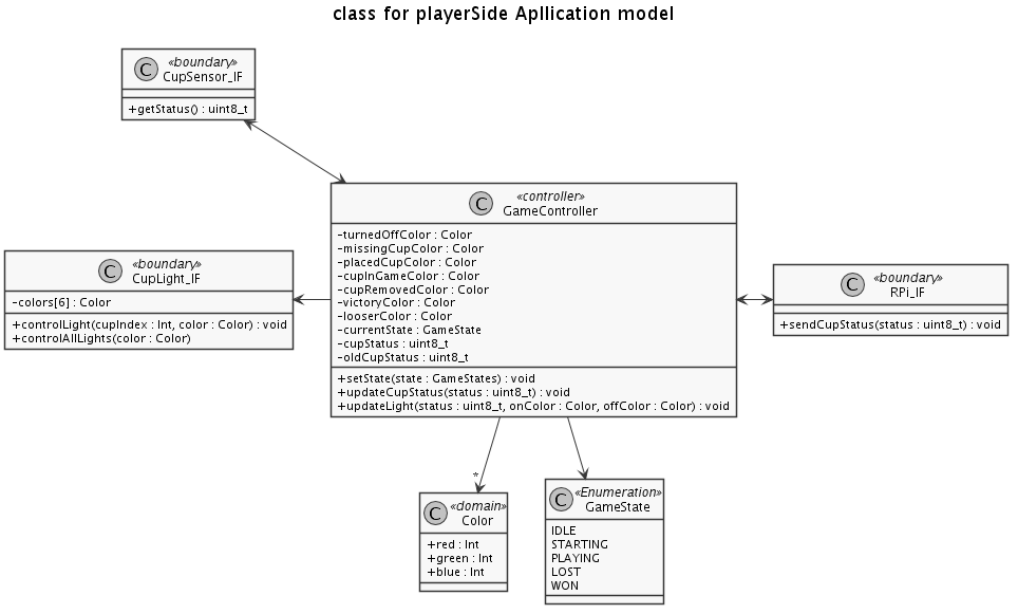
Sekvensdiagram UC4: Bolddispenser

Når der skal udøves service på bolddispenseren, kan dette også beskrives i applikationen som forløbet, der ses i figur 5.13.



Figur 5.13: Sekvensdiagram for UC4 for bolddispenser

5.2.3 playerSideApp



Figur 5.14: Klassediagram for playerSideApp

Controller: GameController
GameController er den centrale klasse for alle use cases. Den sørger for at sende og modtage data gennem klassen I2C. Alt modtaget data bliver evalueret i GameController og sendt videre i overensstemmelse med protokollen og use case flow. Klassen varetager alt logikken og de flest beslutninger i for player side.

Boundary: RPi_IF
Klassen skal modtage kommandoer fra Raspberry pi, og sende data til raspberry pi. Der modtages hovedsageligt data om hvilken tilstand systemet er i og der sendes hovedsageligt koppernes status (om de er der eller ej)

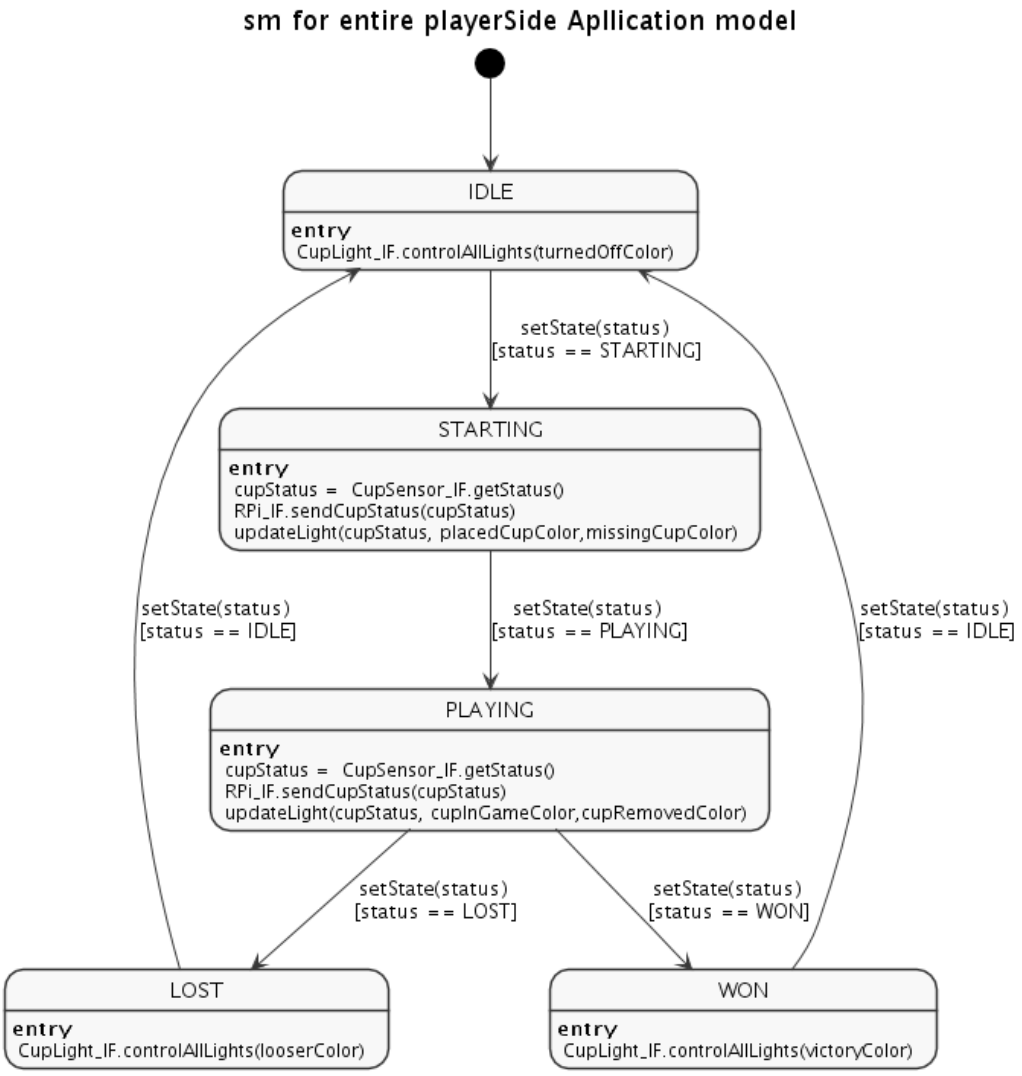
Boundary: CupLight_IF
Klasse som skal styre lyset under hver kop.

Domain: CupSensor_IF
Klasse som skal håndtere signal fra Cup sensorerne. Skal informere GameController klassen når der sker en ændring på status på en af kopperne.

Domain: Color
Klasse som bruges til at lagre oplysninger om de forskellige farver der skal bruges.

Enumeration: GameState
Denne "klasse"er ikke en rigtig klasse. Den bruges til at definere de forskellige tilstande i applicationen.

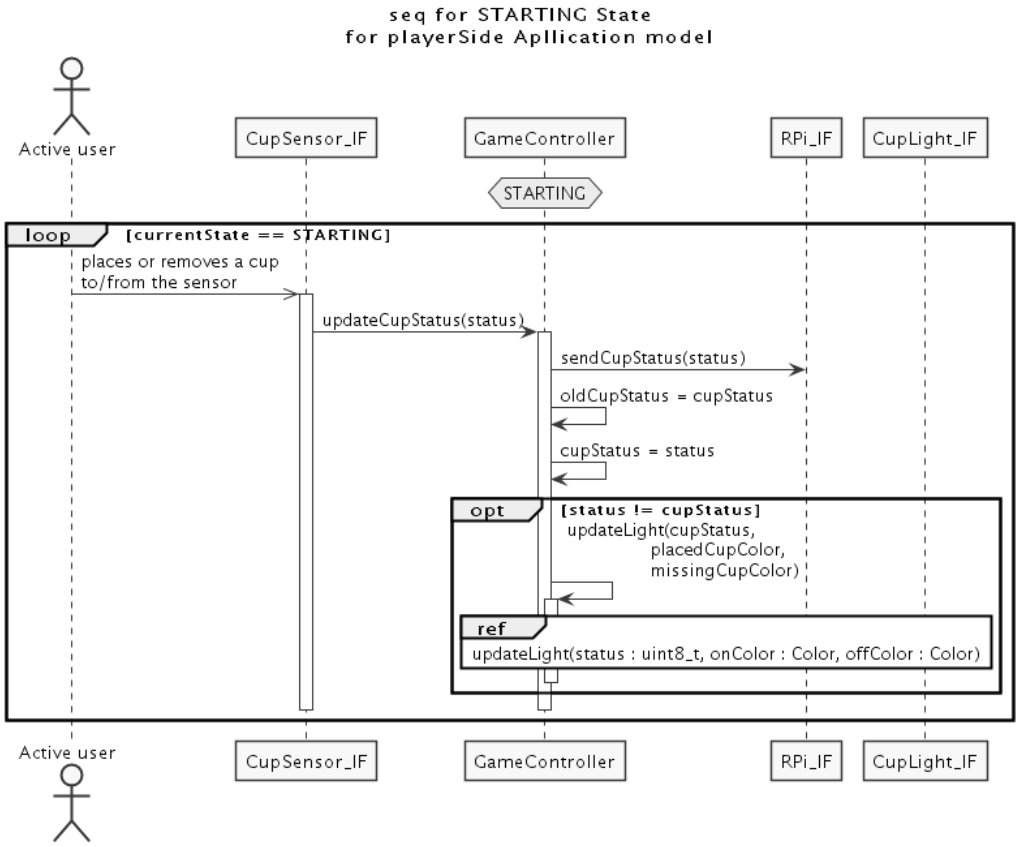
Tilstandsmaskine for playerSideApp
Programmet byges op omkring en tilstandsmaksine, se figur 5.15 . Dette er den vigtigste del af denne applikationsmodel. Tilstandene styres af RPi, og afhængig at hvilken tilstand applikationen er i skal der udføres forskellige ting. Der er de fem tilstande: IDLE, STARTING, PLAYING, LOST og WON. IDLE er tilstanden hvor der ikke er nogen der bruger systemet. Her skal der slukkes for alt lys. Dette gøres ved **entry** for tilstanden. LOST og WON er tilstandene hvor der er fundet en vinder, og den givne side har enten vundet eller tabt. I disse tilstande skal lyset til når man taber eller vinder styres. Når man vinder skal alle kopper lyse med en farve (victoryColor), og når man taber skal de lyse med en anden farve (looserColor). Dette udføres ved **entry** for tilstandene. STARTING er tilstanden hvor spillet opsættes (svare til UC1). Ved tilstanden STARTING skal lyset under hver kop være styret af om der er en kop eller ej. Hvis der er en kop skal den lyse med en farve og hvis der ikke er en kop skal den lyse med en anden. Dette udføres til dels ved **entry** for tilstanden. Men det udføres også hver gang Cup sensor detektere en ændring, se sekvensdiagram for tilstand STARTING. Derudover skal der også sendes information om hvilke kopper der er placeret på bordet (status) til RPi når der sker en ændring, men det sendes også ved **entry**. PLAYING er tilstanden hvor spillet er i gang (svare til UC2). I tilstanden PLAYING skal der stort set ske det samme som i tilstanden STARTING, det er bare nogle andre farver der skal bruges.



Figur 5.15: Tilstandmaskine for playerSideApp

Sekvensdiagram for STARTING

Der laves sekvensdiagrammer for tilstandene STARTING og PLAYING. Der laves ikke sekvensdiagrammer for de andre tilstande, da der ikke skal ske noget i selve tilstanden, der skal kun ske noget ved **entry**, og dette er allerede beskrevet i tilstandsmaskinen. Sekvensdiagrammet for STARTING kan ses på figur 5.16

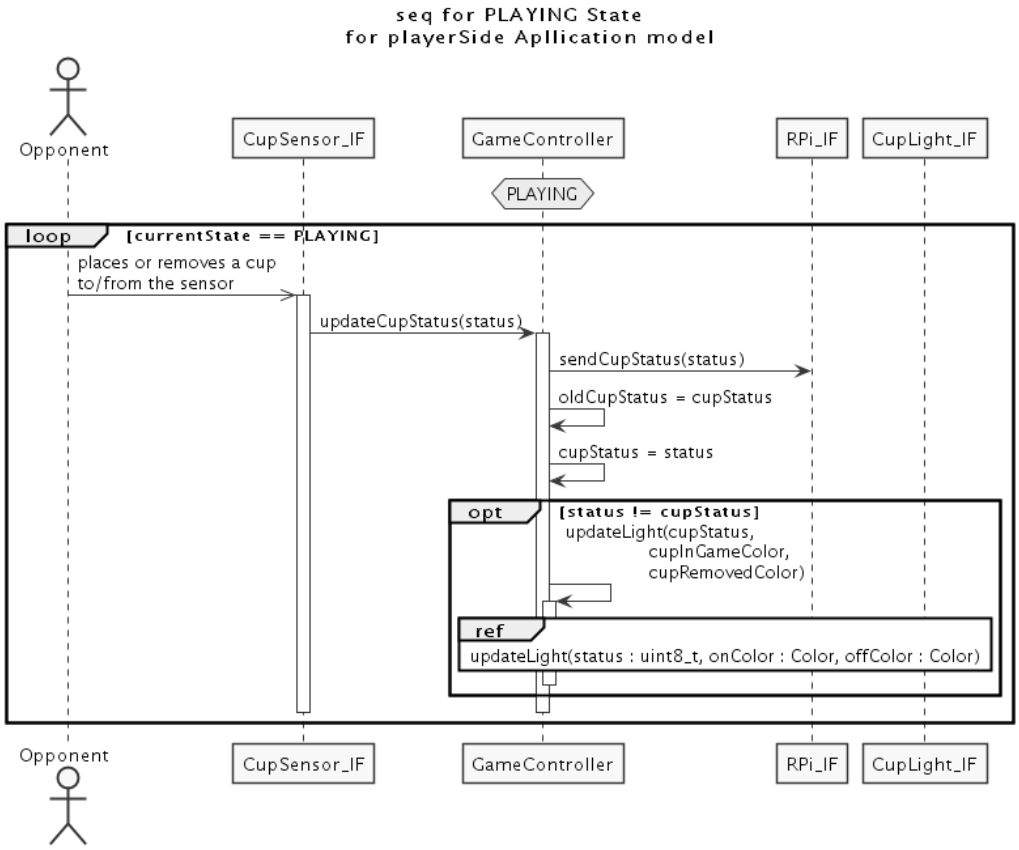


Figur 5.16: Sekvensdiagram for tilstand STARTING

Hver gang der sker tilføjes eller fjernes en kop fra bordet skal denne nye status sendes til GameController og den sørger for at opdatere lysene og sende status til RPi. Der kaldes metoden updateLight som sørger for at indstille hver kop-lys til den angivne farve, afhængig af om der er en kop eller ej. Denne metode er beskrevet i et selvstændigt tilstandsdiagram.

Sekvensdiagram for PLAYING

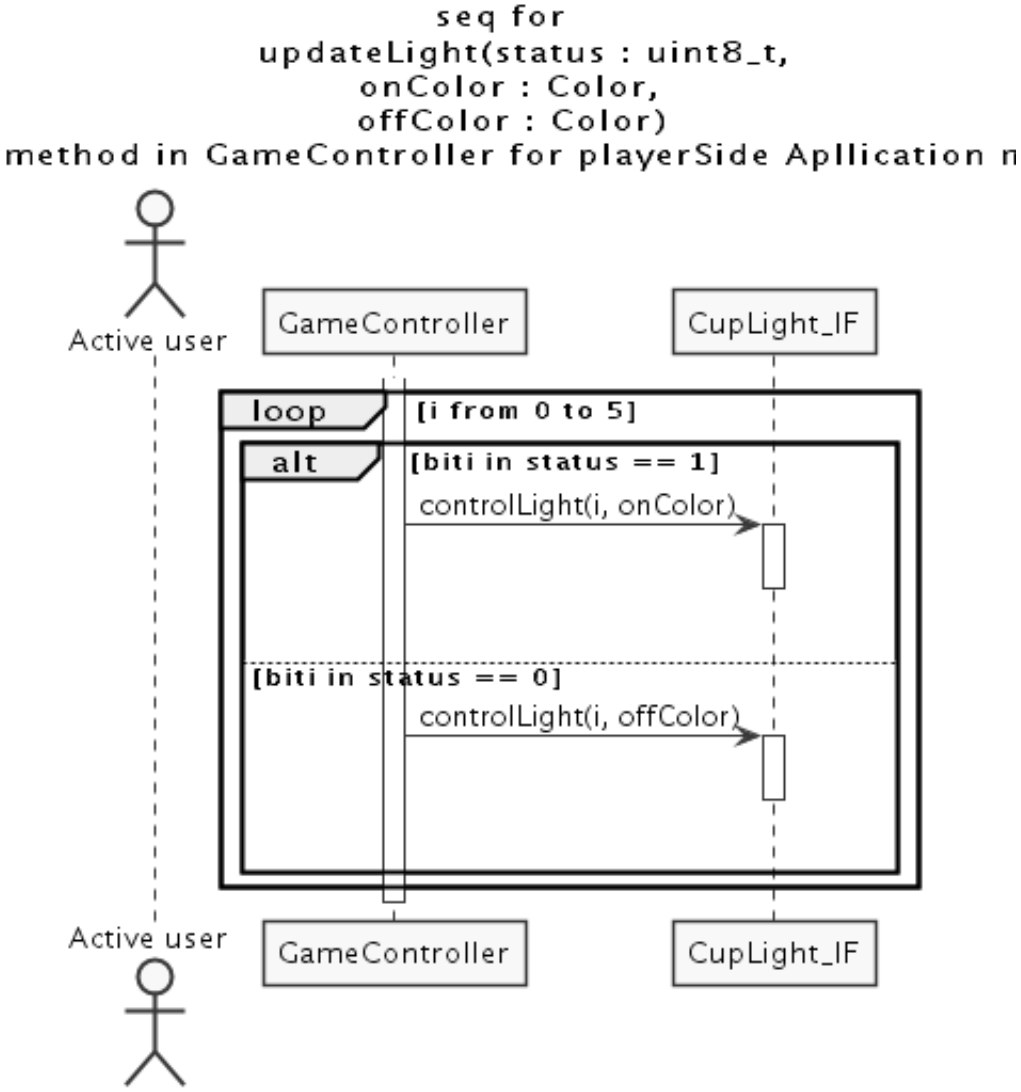
Sekvensdiagrammet for PLAYING kan ses på figur 5.17



Figur 5.17: Sekvensdiagram for tilstand PLAYING

Dette diagram er stort set identisk med det for tilstanden STARTING (figur 5.17) Den eneste forskel er de farver der bruges.

Sekvensdiagram for updateLight metoden
Der er lavet en separat sekvensdiagram for updateLight metoden, se figur 5.18. Denne funktion indstiller lyset for hver kop, afhængig af de angivne parametre.



Figur 5.18: Sekvensdiagram for updateLight