CDIO del 2

02312, 02313, 02315

S182946
Marah Osama Marak

S195457
Jonas Løvenhardt Henriksen

S195470
Mikkel Hillebrandt Thorsager

S195379
David Hoffmann Jeppesen

S195453
Sander Eg Albeck Johansen

S181528
Madeleine Glindorf

Indledende programmering, Udviklingsmetoder til IT-systemer og Versionsstyring og Testmetoder

Timeregnskab

	21/10	22/10	28/10	29/10	1/11
Marah					1 time
Madeleine					
Sander	0.5 time	1 time	2 timer	3 timer	4 timer
David			1.5 timer	1.5 timer	2 timer
Mikkel				1 time	4 timer
Jonas	0.5 time	0.5 time	1.5 time	1.5 time	4 timer

Resume

I indledningen fik vi etableret formålet med denne opgave baseret på flere forskellige kurser.

I krav delen gik vi i dybden med de krav og den vision der er til vores projekt, Herefter, satte vi analyse op af de navneord som ville spille en rolle i projektet. Efterfulgt af en analyse af de ikke-funktionelle krav samt de funktionelle krav som bliver stillet til vores projekt, efterfulgt af en domænemodel som illustrere hvordan vores kode fungere.

I analysedelen startede vi med at lave stille alle vores use cases op efterfulgt af en fully dressed use case analyse.

I design delen har vi lavet diverse diagram for at hjælpe os med bedre at designe vores program på en overskuelig måde. Vi har startet med at lave et systemsekvensdiagram som illustrere den rækkefølge vores program kommer til at fungere i for en given use case. Herefter har vi lavet et sekvensdiagram som illustrere hvordan vores metoder arbejder sammen. Til slut har vi lavet et designklassediagram som illustrerer hvordan vores klasser interagere med hinanden.

I implementerings delen har vi arbejdet med at teste programmet og beskrevet hvordan programmet fungere.

I projektforløbet delen har vi beskrevet hvordan at selve projektet forløb og hvordan vi har samarbejdet om at løse den givne opgave.

Til sidst har vi konkluderet baseret på det formål som vi beskrev i indledningen og kommet frem at formålet er blevet opfyldt.

Indledning

I denne opgave skal vi udvikle et spil for IOOuterActive. Det skal være et spil der forgår mellem to spillerer med et raflebæger med to seks sidede terninger som skal kunne spilles på maskinerne i DTU´s databarer. Spillet skal gå ud på at man starter med 1000 guldmønter, og man skal være den der først får 3000 guldmønter. Det gøres ved at man slår med terninger og udfra summen af dem, lander man på et felt, hvor man får en tekst, hvor mange points man får eller mister, og om man må slå igen. Vi kommer også ind på Analyse, design, konfiguration og test af vores program.

Kundens vision og ønsker

Vi blev imponeret over jeres terningespil og vil derfor gerne have at I udvikler et nyt spil til os. Ligesom tidligere skal det være et spil mellem 2 personer, der kan spilles på maskinerne i DTU's databarer, uden bemærkelsesværdige forsinkelser.

Spillerne slår på skift med 2 terninger og lander på et felt med numrene fra 2 - 12. At lande på hvert af disse felter har en positiv eller negativ effekt på spillernes pengebeholdning. (Se den følgende feltoversigt), derudover udskrives en tekst omhandlende det aktuelle felt. Når en spiller lander på Goldmine kan der f.eks. udskrives: "Du har fundet guld i bjergene og sælger det for 650, du er rig!".

Spillerne starter med en pengebeholdning på 1000. Spillet er slut når en spiller når 3000. Spillet skal let kunne oversættes til andre sprog. Det skal være let at skifte til andre terninger.

Vi vil gerne have at I holder jer for øje at vi gerne vil kunne bruge spilleren og hans pengebeholdning i andre spil.

Feltliste

1.	(Man kan ikke slå 1 med to terninger)	
2.	Tower	+250
3.	Crater	-100
4.	Palace gates	+100
5.	Cold Desert	-20
6.	Walled city	+180
7.	Monastery	0
8.	Black cave	-70
9.	Huts in the mountain	+60
10.	The Werewall (werewolf-wall)	-80, men spilleren får en ekstra tur
11.	The pit	-50
12.	Goldmine	+650

Analyse

Non funktionelle krav

Da programmet ikke er særligt stort er det nemt og hurtigt at vedligeholde. Dette medfører også, at det er nemt at lave ønskede ændringer i, samt at se hvilke funktioner de forskellige klasser i koden har, som igen så fungerer hurtigt. Vi har fjernet næsten al form for user input, da det eneste brugerne kan gøre er at skrive deres navne og trykke enter. Dette medfører, at vi fjerner en risiko for at brugeren kan begå en fejl, som gør det markant mere brugervenligt og stabilt.[1]

Funktionelle krav

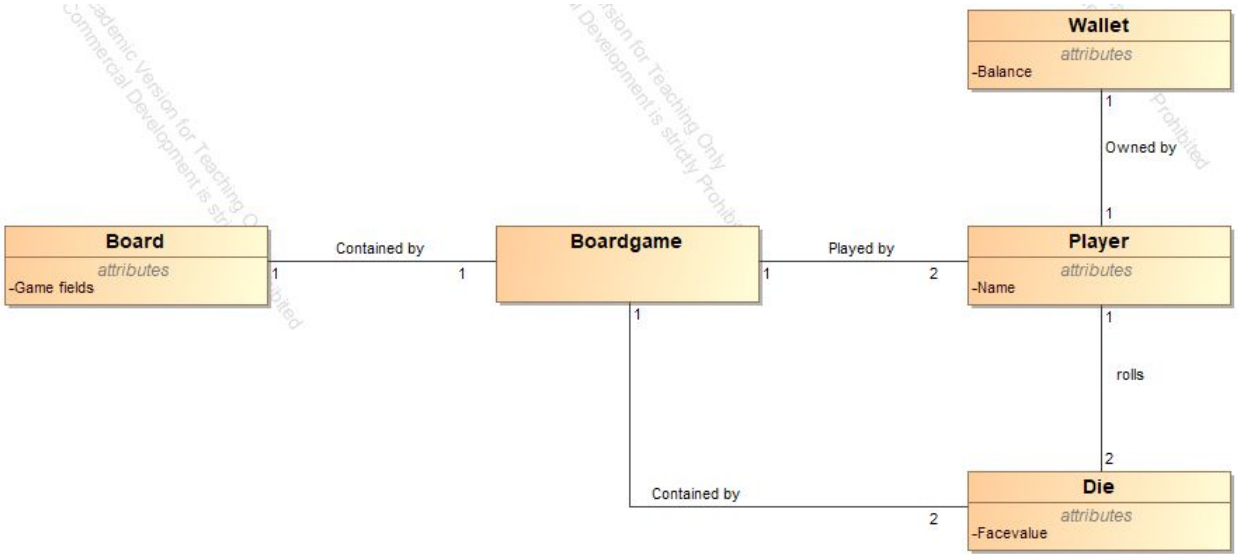
1. To spillere skiftes til at slå
2. Man lander på felter med numre fra 2-12
3. Hvert felt har en effekt på spillerens pengebeholdning
    a. Felt 10 giver spilleren en ekstra tur
4. Hver spiller starter med 1000 guldmønter
5. Spillet er slut, når en spiller har nået 3000 guldmønter
    a. Vinderen er den spiller, der først opnår en pengebeholdning på 3000 guldmønter
6. Spillet skal kunne oversættes til andre sprog
7. Det skal være nemt at bruge nogle andre terninger
8. Spilleren indtaster selv navn
9. Spilleren slår selv terningerne via user input
10. Mulighed for at bruge spilleren og pengebeholdningen i andre spil

[1] https://en.wikipedia.org/wiki/FURPS

Navneordsanalyse

- Spil,
- Spiller1
- Spiller2
- Terning,
- Felt
- Tur
- Guldmønter
- Pengebeholdning
- Navn
- Effekt
- Terningekast

Domænemodel



Use cases

U1: Begge spillere indtaster deres navne

U2: Spillet kan se hvis tur det er

U3: Slå med de 2 terninger

U4: lande på et felt ud fra hvad terninger lander på, som så vil have en effekt på din pengebeholdning

4A: Hvis der landes på felt nr. 10 skal spilleren slå igen

U5: At tjekke penge beholdningen

U6: Når en spiller har nået 3000 kr er spillet slut og spilleren med 3000 kr har vundet

Fully dressed use case:

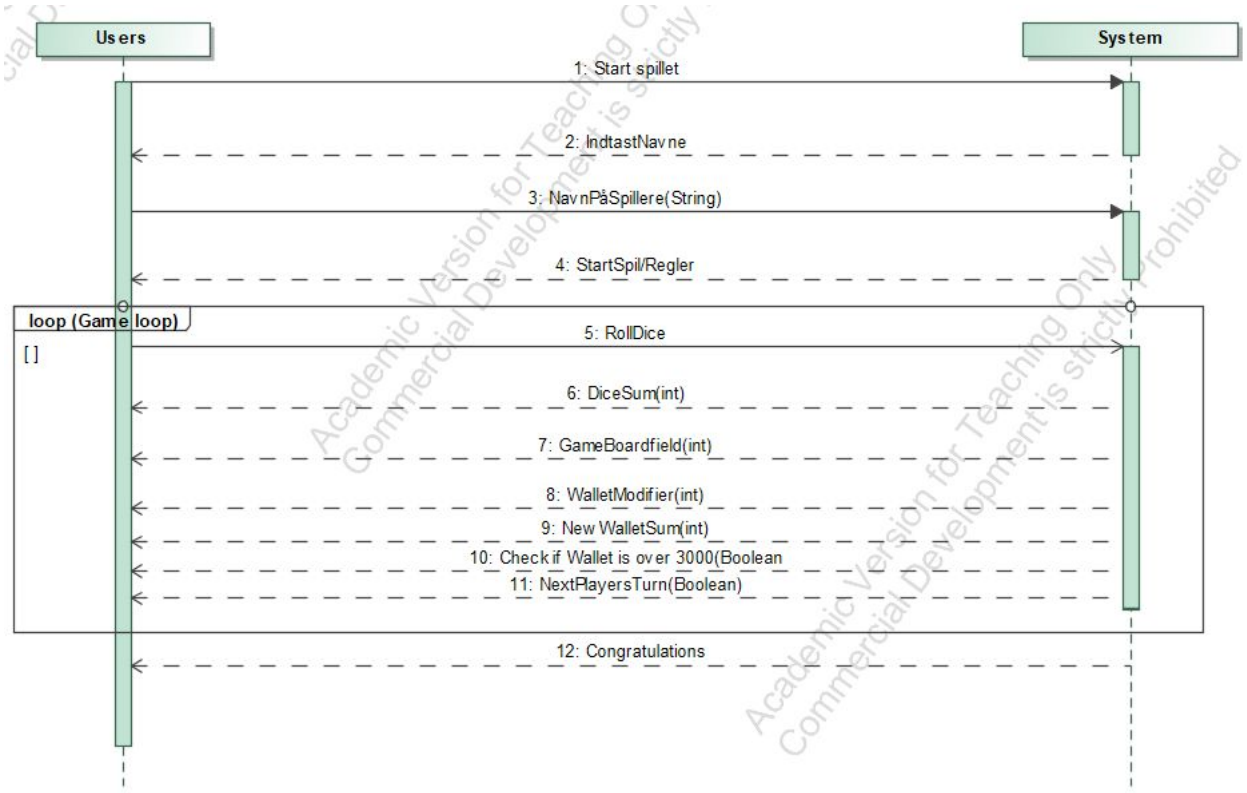
Use case navn	Gameboard interaktioner
ID	U4
Level	For hvert terningkast lander spilleren på et felt som har en bestemt funktion.
Primære aktører	Spilleren og systemet.
Andre interessante	Kunden som forventer at systemet virker. Udvikleren i form af at besvarelsen skal bruges i eksamen, og udvikler håber derfor at besvarelsen er fyldestgørende.
Forudsætninger	Spillet skal være startet, dvs spillerne skal have indtastet deres navne. Terningerne skal være slået.
Succeskriterier	Når terningerne er slået lander spilleren på et felt, informationen fra det felt gives videre til brugeren.
Basic Flow	1. Navnene på spilleren skal være indtastet 2. Terningerne bliver slået 3. Spiller lander på et felt alt efter udfaldet af terningekastet 3a hvis terningerne viser 2 får spilleren 250 mønter 3b hvis terningerne viser 3 mister spilleren 100 mønter 3c if terningerne viser 4 får spiller 100 mønter 3d hvis terningerne viser 5 mister spiller 20 mønter 3e hvis terningerne viser 6 får spiller 180 mønter 3f hvis terningerne viser 7 sker intet og turen går videre 3g hvis terningerne viser 8 mister spiller 70 mønter 3h hvis terningerne viser 9 får spiller 60 mønter 3i hvis terningerne viser 10 mister 80 mønter men får en ekstra tur 3j hvis terningerne viser 11 mister spiller 50 mønter 3k hvis terningerne viser får spiller 650 mønter 4. Efter hvert terningekast går turen videre til næste spiller (medmindre udfald 3i opstår) 5. Når en spiller har nået 3000 mønter er spillet slut og vinderen er spilleren med mindst 3000 mønter

Design

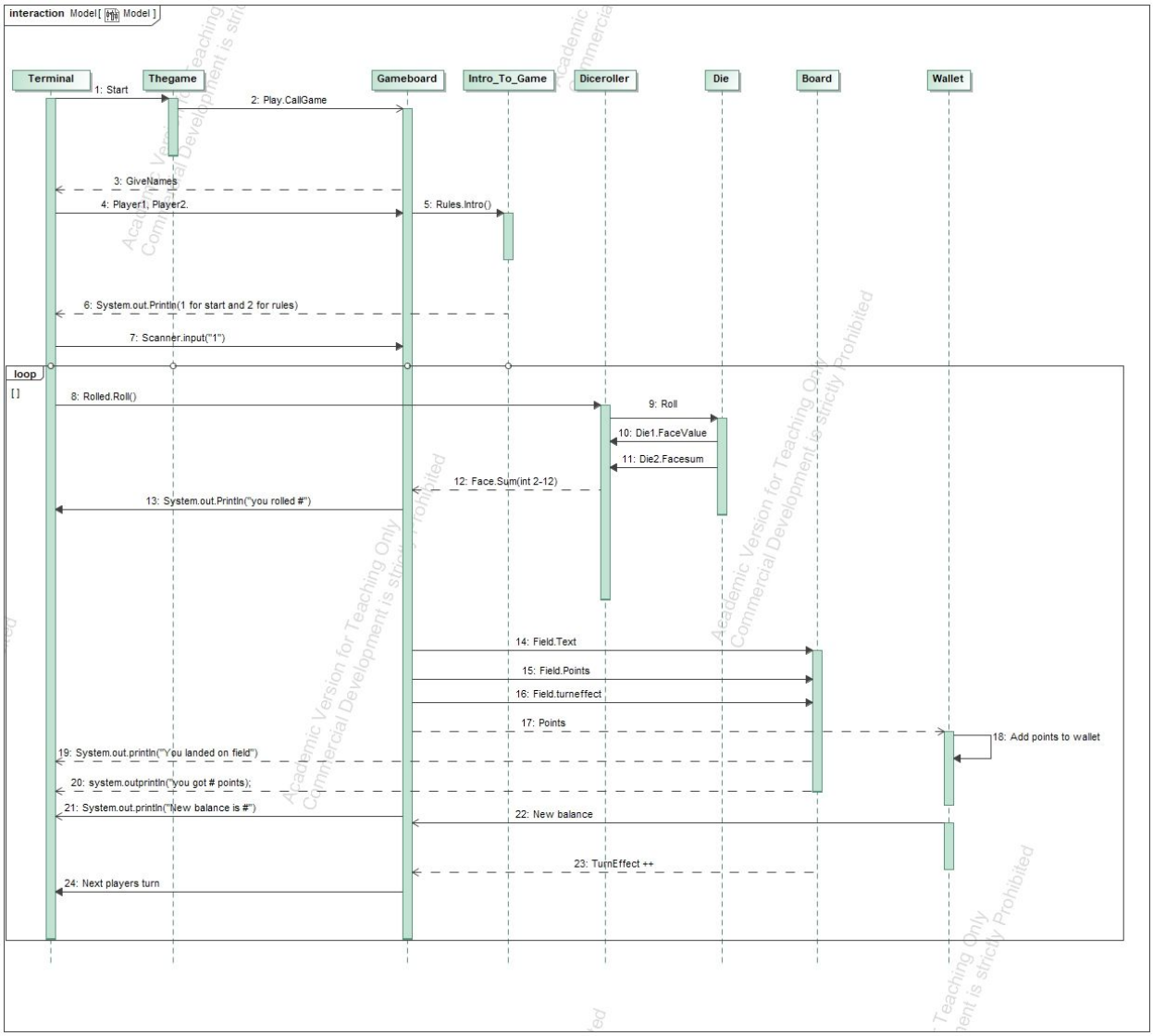
Vores program indeholder to pakker, en main pakke “com.company” og en pakke “Tests” med test klasserne.

Test	com.company
Dietest (Tester om terningen slår mellem 1 og 6)	Board (Indeholder tekst der skal skrives, hvor mange mønter et felt giver eller tager og om spilleren skal have en ekstra tur.)
Dicecuptest (Tester spredningen af terningsslagene)	Dicecup (Kalder terningerne til at rulle og lægger deres værdier sammen.)
Wallettest (Tester at balacen ikke kan komme under 0)	Die (Indeholder en terning, som returneres med en tilfældig værdi ml. 1 og 6)
	Gamelogic (Indeholder hvad der sker efter hver spiller har slået samt en winning-condition)
	Intro_to_game (Indeholder introen til spillet, hvor der kan besluttes om man ønsker reglerne eller ej)
	Main (Starter spillet)
	Player (Indeholder spillerne og tilknytter en wallet til begge spillere)
	Rules (Indeholder reglerne for spillet som kan kaldes hvis det ønskes)
	Wallet (Indeholderen Wallet’en, hvor der enten kan lægges til eller trækkes fra)

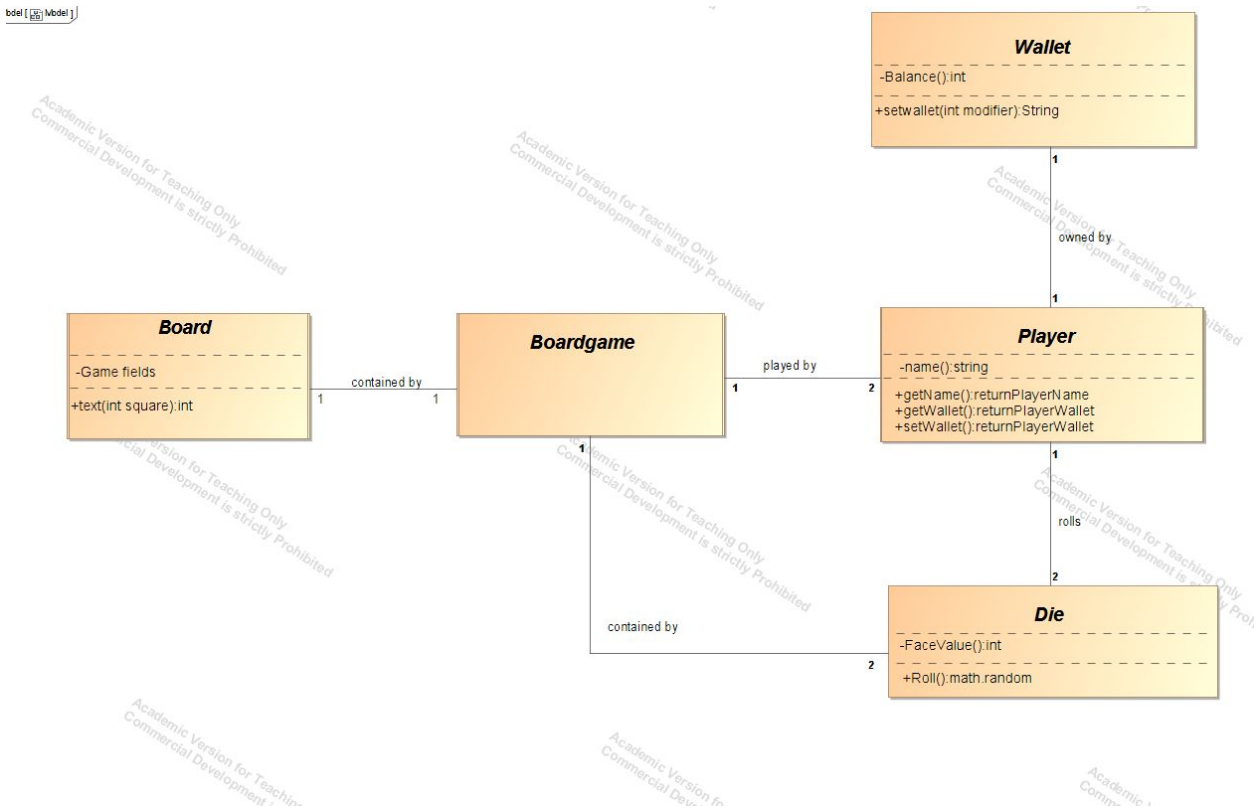
System sekvensdiagram



Sekvensdiagram



Design-klassediagram



Konfiguration

Styresystem - Dit styresystem skal bare kunne køre IntelliJ IDEA 2019.2.1

Installeret programmer - IntelliJ IDEA 2019.2.1, jdk 13.0.1

Minimumskrav - Et keyboard eller lignende om kan lave inputs

Hvordan koden hentes fra et git-repository - Brug linket under “Git-Hub” → Tryk “Clone or Download” → unzip filen i en mappe

Hvordan kildekoden skal compiles, installeres og afvikles - åben IntelliJ IDEA → vælg mappen hvor man har unzippet filen → Åben klassen "Main.java" → Byg og kør programmet!
(jdk version 13.0.1 skal måske vælges, hvis IntelliJ ikke allerede bruger den som standard

Test

For at teste vores program, har vi set på hvilke af vores klasser der giver mening at teste. Vi er derfor kommet frem til at vi vil teste klasserne wallet, die og dicecup. Til testene har vi set på hvilke situationer der kan opstå, og set op programmet kan håndtere dem.

Til testen af dicecup har vi skrevet et kodestykke til at rulle terningerne 10.000 gange, og set hvordan spredningen af summen af terningernes ansigter. Vi har herved fået følgende spredning

```
2: 290 3: 563 4: 844 5: 1064 6: 1447 7: 1643 8: 1399 9: 1081 10: 818 11: 584 12: 267
```

Den spredning passer meget godt overens med, at sandsynligheden for at slå de midterste værdier er størst. Vi konkluderer herfra at vores klassen dicecup give et godt billede af at kaste med to virkelige terninger.

Til at testen klassen die har vi set om terningen viser en værdi mellem 1 og 6. For at teste terningen har vi brugt junit test. Testen ruller terningen 1000 gang og evaluerer for hver gang om terningen viser imellem 1 og 6.

Til testen af wallet klassen har vi set på hvilke mulige situationer der kan opstå. Vi har så ud fra disse muligheder sat test op til at teste dem. En mulighed kunne være at der er x penge på kontoen og der skal trækkes et beløb større end x fra kontoen. I testen vælges så et beløb større end kontoens saldo, vi kan hermed se om kontoen går under nul.

```
@Test
public void setWallet1() {
    Wallettest.SetWallet( modifier: -1100);
    assertEquals(Wallettest.GetWallet(),  actual: 0);
}
```

Vores test viser at det gør den ikke, og vi ser også fra testen af de andre muligheder, at klassen wallet sætter saldoen til den forventede værdi.

Projektforløb

Vi startede ud med at sætte et GITHUB respiratory op, så gruppen kunne begynde at eksperimentere med det vi havde lært i 02312. Vi har mødtes på fastlagte datoer og har aftalt at bruge unified proces, vores arbejde er derfor blevet langt mere struktureret end sidst.

Konklusion

Det er lykkedes os at få koden til at virke og spillet fungere derfor perfekt. Vi har været langt bedre til at planlægge og holde orden i forhold til cdio 1, hvad der jo var vores mål. Vores udviklingsplan har fungeret via unified process som har fungeret rigtigt godt for vores arbejde.

Git-Hub

https://github.com/SanderJohansen/CDIO2