Danmarks Tekniske Universitet

CDIO 1

Gruppe 18:
Ali Harb El-Haj Moussa (s175119)
Magnus Hansen (s175198)
Niklaes Dino Jacobsen (s160198)
Oliver Storm Køppen (s175108)
Sebastian Bilde (s175116)

Underviser:
Ian Bridgwood, 02313
Stig Høgh, 02314
Henrik Tange, 02315

24. oktober 2017

Gruppens medlemmer

Ali Harb El-Haj Moussa (s175119)

Magnus Hansen (s175198)

Niklaes Dino Jacobsen (s160198)

Oliver Storm Køppen (s175108)

Sebastian Bilde (s175116)

Ekstra informationer

Projekt på Github: `https://github.com/group-18/CDIO1`

Abstract

We intended to create a program, that can simulate a dicegame between two players. There are two dices, and certain dice rolls, that reacts in different ways. Each player gets points according to the facevalue.
The main goal is to reach 40 points, to win the game.
The program is controlled by the user in a console-enviroment.
The dicegame is written in the programlanguage Java.
Our test and improvements on the program makes the dicegame as random as possible, while still following the rules. Our program is functioning as predicted, and is reaching the requirements of our project.

Indhold

1 | Indledning

I denne rapport vil vi gennemgå en opgave, som spilfirmaet *IOOuterActive* har modtaget, og vi, som udviklere for dette firma, vil skabe dette spil, som opgaven beskriver. Opgaven vil i denne rapport optræde under navnet 'CDIO1'.

CDIO1 er et projekt, der går ud på at skrive et terningespil. Vi har i denne opgave forsøgt, at arbejde på en agil måde, og har derved valgt at arbejde med UP (Unified Process). UP går i sin helhed ud på, at dele projektet op i iterationer. Normalt vil disse vare 2-6 uger, men i dette korte projekt har vi ikke tidsbestemte iterationer. I projektet har vi primært brugt brugt følgende software:

1. **Visual Studio Code**
   Vi valgte fra starten at skrive i LaTeX, for at lette skrivningsprocessen og få en homogen rapport. Samtidig er filtypen .tex understøttet af Git, til versionsstyring.

2. **Tower**
   Tower er et af de mange GIT-UI på markedet, og det fungerer eminent, til de funktioner, vi har benyttet os af.

3. **IntelliJ**
   Vores program er skrevet i IntelliJ, sideløbende med Eclipse for at teste, om slutproduktet fungerer for flere forskellige maskiner og i forskellige programmer.

4. **Asana**
   Asana er et projektstyringsprogram, hvor man kan skabe opgaver og tildele personer til opgaver med eventuelle deadlines.

5. **Slack**
   Slack er en *digital workspace*, hvor man kan kommunikere med projekts medarbejdere.

2 | Problemformulering

Er det muligt at producere et spil, som består af to spillere, der hver slår med to statisk *ærlige*[1] terninger?

Spillets regler:

Spillet går ud på, at hver spiller på skift, skal kaste med to terninger, for at nå en sum på 40 point. Den første spiller, der slår to ens terninger, mens spilleren har 40 point eller derover har vundet.
Hvis en spiller slår to ens, så får personen en ekstra tur. Det gælder dog, at hvis en spiller slårs to 6'ere i to runder, så har spilleren vundet uanset point. Hvis spilleren slår to 1'ere, så vil spillerens miste alle sine point.

[1] En ærlig terning en sandsynlighedsfordeling, hvor alle udfald har lige stor sandsynlighed

3 | Analyse

I dette kapitel vil vi gennemgå kravspecifikation til programmet, samt optegne og forklare forskellige modeller.

3.1 Kravspecifikation

1. Spillet skal håndtere to spillere.
2. Man skal slå med to terninger, som begge er fair.
3. Summen af de to terningerner ligges sammen og gives til spillerens pointsum.
4. Man vinder ved at nå 40 point.
5. Ved at slå to 1'ere, nulstilles spillerens point.
6. To ens giver ekstra tur.
7. Skal kunne huske spillerens forrige kast.
8. Spilleren kan vinde med at slå to 6'ere i forrige kast og nuværende kast.
9. To ens for at vinde spillet efter 40 point.

3.2 Interessentanalyse

Interresent	**Interesse / Mål**
Spiller/-e	Kunne styre et system, der styrer et spil mellem 2 personer, hvor i der kastes med et raflebæger med to terninger i og ser resultatet med det samme.

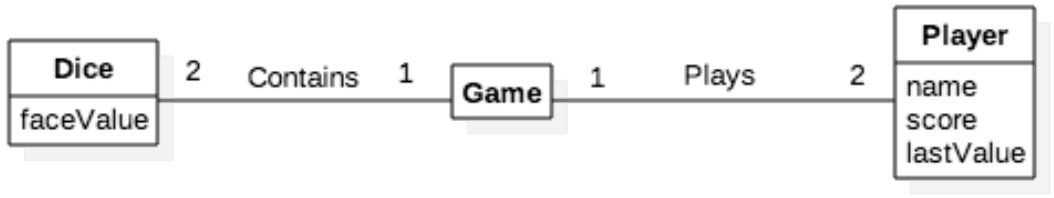
3.3 Use cases

Use case: PlayGame
ID: 1
Brief description Spiller/-e skal kunne spille spil/Game (starte spillet) og slå med terningen
Primary actors: Spiller/-e
Secondary actors: Ingen.
Preconditions: Ingen.
Main flow: 1. **Spiller 1 indtaster navn** 2. **Spiller 2 indtaster navn** 3. **Spiller 1 indtaster 'roll' for at slå med terningen** 4. **Spiller 2 indtaster 'roll' for at slå med terningen** 5. **Opnås 40 point, og to ens terninger slås, stoppes spillet**
Postconditions: Ingen.
Alternative flow: - Ekstra ture gives ved, at slå to ens terninger - Slås to ens, med 1'ere bliver spillerens score nulstillet - Spil afsluttes, hvis der bliver slået to 6'ere, med den forudsætning, at to 6'ere blev slået i den forrige runde

Tabel 3.1: Use case 1

3.4 Domænemodel

For bedre at kunne forstå domænet omkring spillet, har vi ud fra use case 1 (Tabel 3.1) lavet en domænemodel. domænemodellen beskriver de elementer vi vil arbejde med. Det ses der, at spillet kommer til at bestå af terningerne og spillere. På figur 3.1 ses domænet. Der ses at både terninger (Dice) og spillere (Player) refererer til spillet (Game).

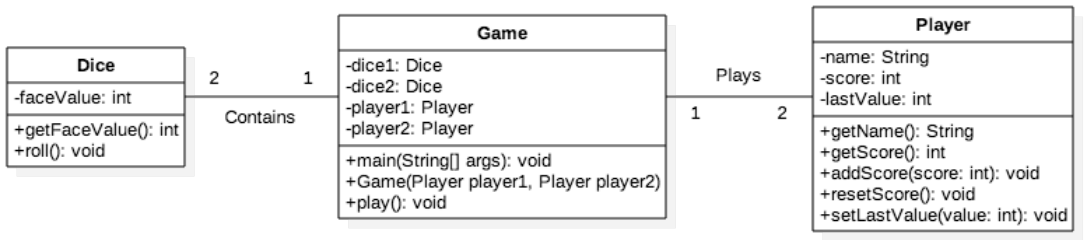


Figur 3.1: Domænemodel over spillet

4 | Design

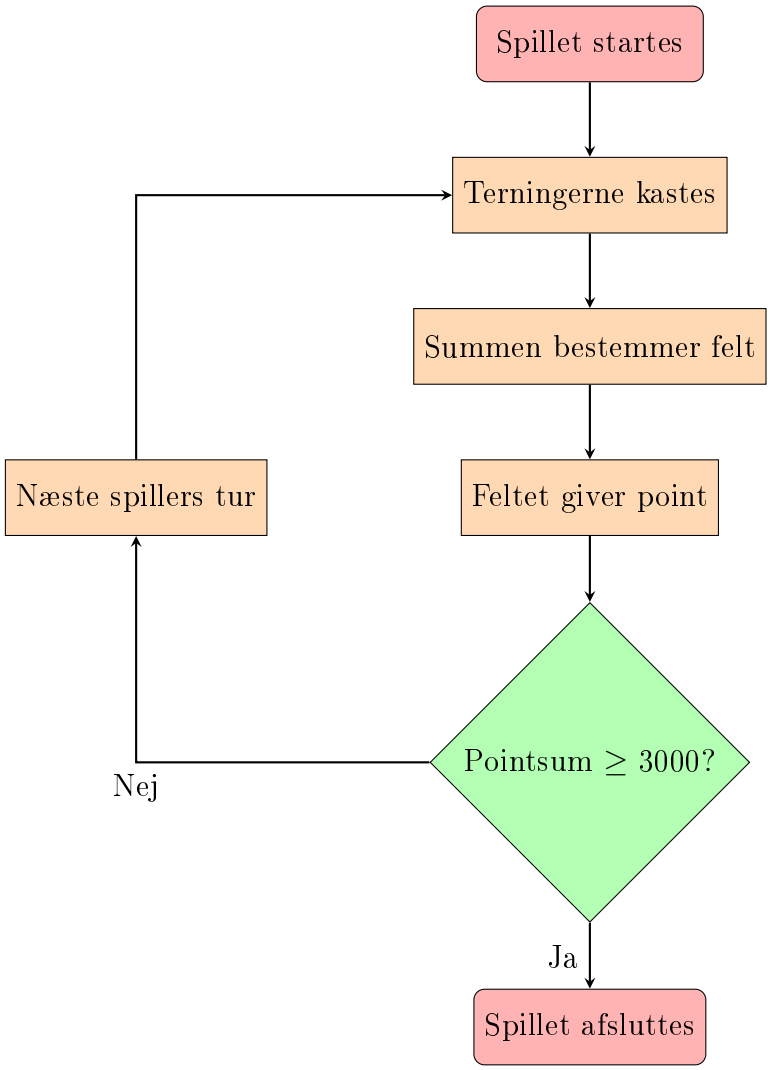
For bedre at forstå designet omkring spillet, har vi ud fra use case 1 (Tabel 3.1), lavet et klassediagram som beskriver de elementer spillet består af. Det ses der, at spillet kommer til at bestå af terningerne og spillere. På figur 4.1 ses klassediagrammet.

4.1 Klassediagram



Figur 4.1: Klassediagram over spillet

4.2 Flowdiagram



5 | Implementering

I vores arbejde med at kode dette program, har vi startet ud med at se bort fra de ekstra krav vi har fået, og blot fået implementeret kerneprogrammet.
Kerneprogrammet er programmet, der opretter to spillere, og to terninger, og den spiller der først når 40 point vinder. Vi besluttede os for at lave et while loop, således at programmet fortsætter med at køre, indtil vi har opnået en bestemt score.
Dermed er kerne-delen af programmet blevet lavet. Herefter begyndte vi på, at arbejde med, at implementere de ekstra krav, der er stillet. Dette blev grundigt overvejet, før valgtet af form for kode, blev bestemt.
Vi fandt ud af, at alle ekstra krav ville blive løst, ved at benytte if-else sætninger, hvor i krav kriterierne indgår. De første 3 krav var relativt enkle at opnå, da disse bare bestod af simple if-else sætninger. Det fjerde krav, var sværere at få implementeret. Kravet lyder på, at man kun kan vinde på et par. Vi har valgt at kode programmet sådan, så hvis man får over 40 point, men ikke slår en double, vil programmet sætte din score til 40. Dette har vi gjort, da det er ligegyldigt om man har 40 eller 100 point, ligeså snart man har 40 point skal man bruge et par for at vinde. Ved at stille det op på denne måde, sørger vi for, at while loopet fortsætter, til at der er en spiller der både har en score>=40, og at han har slået et par i denne runde.

5.1 Brugervejledning for programmet

Programmet er meget simpelt sat op, og har en masse System.out.println der fortæller brugeren præcis hvad der foregår, og hvad han skal gøre for at programmet fortsætter.
Programmet starter med at printe alle spillets regler, således at brugeren kan se reglerne. Derefter beder programmet brugeren om at indtaste navn på spiller 1, og navn på spiller 2. Herefter vil programmet blive ved med at spørge brugeren, om at skrive "roll", indtil brugeren taster dette. Programmet vil herefter printe et terningeslag. Summen af slaget, og stillingen mellem de to spillere, i tilfælde af at der er slået et par, vil programmet skrive at det er den samme spillers tur. Programmet bliver ved med dette, indtil en spiller opnår over 40 point, efter dette vil programmet skrive, at man skal slå et par for at vinde. Dette bliver den ved med indtil at en af spillerne slår et par, der ikke er par 1, da par 1 selvfølgelig nulstiller spillerens score.

6 | Test

I dette afsnit vil vi forklare, hvordan vi har lavet test af vores software. Vi har blandt andet gjort brug af unit tests og test cases.

6.1 Unit test

For at softwaren bag holder en vis kvalitet, er der udført en generel unit test på klasserne i spillet. Denne unit test indebærer simple test af alle metoder i klasserne. Derudover er lavet to udvidede test.
Den ene, tester om terningen er fair, med 60000 kast og en fejlmargen på 400 slag pr. side af terningen. Den anden, kigger på, om sandsynlighederne for de respektive summe, ved to terninger er sandsynligt korrekte. Dette gøres også ved 60000 kast, med 2 terninger, for derefter at beregne sandsynligheden, for den respektive sum, samt det faktiske resultat. Nedenfor ses det udsnit af testen, som ordner denne beregning.

```
1  double oneRollPercent = (double) 100 / numberOfRolls;
2
3  double totalPercentage = 0;
4  double failureRate = 0.75;
5  for (int i = numberOfDies; i < rolls.length; i++) {
6      double percentage = (double) rolls[i] * oneRollPercent;
7      int posibilities = this.numPossibilities(numberOfDies, i);
8      double expectedPercentage = posibilities * (100.0 / 36.0);
9
10     totalPercentage += percentage;
11 }
```

6.2 Testcases

Herunder er en række testcases, som er blevet udarbejdet ud fra vores krav og mål. Alle test er blevet testet i IntelliJ.

Testcase 1 handler om, hvorvidt programmet tager korrekt imod Strings, når den beder om navne, og hvorvidt det bliver gemt. Det ville blive kaotisk, hvis der var blevet sat et navn til f.eks. Player1, og programmet så senere omtaler Player1 som Player1 og ikke det pågældende navn.

Test case ID	TC01
Summary	Teste, om programmet modtager navne korrekt
Requirements	
Preconditions	
Postconditions	Programmet kører stadig
Test procedure	1. Indtast namePlayer1 2. Indtast namePlayer2
Test data	Player 1 = Sebastian Player 2 = Magnus
Expected result	Navnene erstatter Player1 og Player2
Actual result	Navnene erstattede Player1 og Player2
Status	Godkendt
Tested by	Oliver Køppen
Data	12/10-2017
Test environment	IntelliJ 2017.2.5 on Windows

Testcase 2 handler om, hvorvidt programmet kan tage imod to ens terninger og ikke give turen videre.

Test case ID	TC02
Summary	Teste, om spillet giver spilleren en ekstra tur, hvis der bliver slået to ens
Requirements	
Preconditions	Spilleren har slået to ens
Postconditions	Spilleren får en ekstra tur
Test procedure	1. Player1 skal taste 'roll' for at kaste med terningerne. 2. Player1 slår to ens og næste tur skal derfor være Player1
Test data	1. Kast 1 : d1=3, d2=3 2. Kast 2: d1=3, d2=1
Expected result	Player1 sum = 10
Actual result	Player1 sum = 10
Status	Godkendt
Tested by	Oliver Køppen
Data	12/10-2017
Test environment	IntelliJ 2017.2.5 on Windows

Testcase 3 handler om, hvorvidt programmet husker det foregående kast, og om spilleren vinder, hvis personen slår to 6'ere på to kast.

Test case ID	TC03
Summary	Teste, om man vinder, hvis man slår 2x6 i to runder
Requirements	TC02 skal gøre sig gældende
Preconditions	Spilleren har slået d1=6 og d2=6 i forrige runde
Postconditions	Spilleren vinder spillet
Test procedure	1. Spiller1 slår sum=12 2. Spiller1 slår sum=12 i anden runde
Test data	1. Kast 1: d1=6, d2=6 2. Kast 2: d1=6, d2=6
Expected result	Vinderskærmen vises
Actual result	Vinderskærmen vises
Status	Godkendt
Tested by	Oliver Køppen
Data	12/10-2017
Test environment	IntelliJ 2017.2.5 on Windows

Testcase 4 handler om, hvorvidt programmet resetter spillerens score, hvis personen slår to 1'ere.

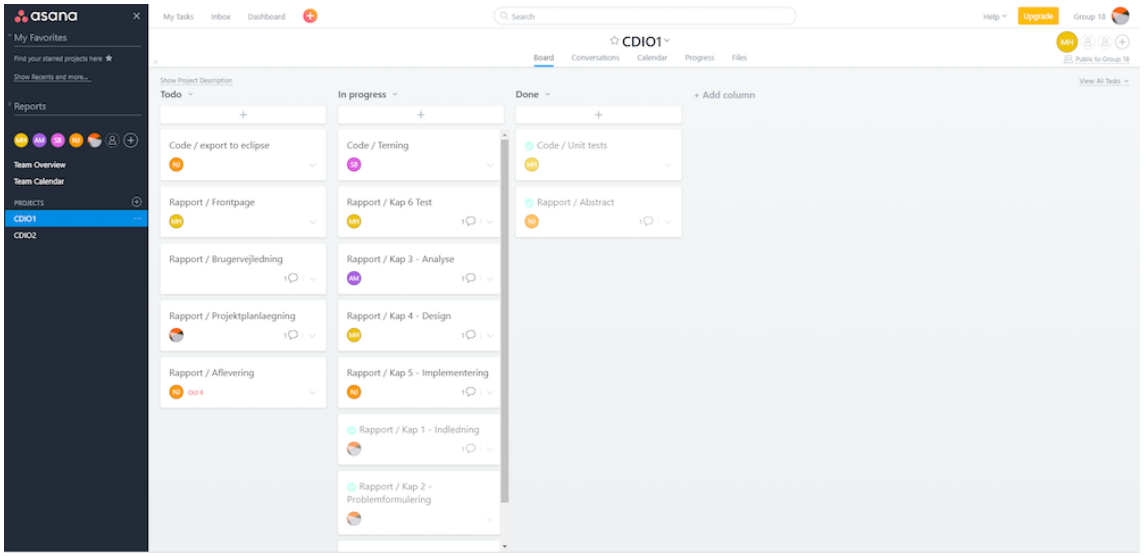
Test case ID	TC04
Summary	Teste, om scoren resettes, når man slår 2x1
Requirements	
Preconditions	Spilleren slår d1=1 og d2=1
Postconditions	TC02 gør sig gældende, og spilleren vil miste sine point, men får en til tur
Test procedure	1. Spiller1 slår sum=2 2. Spiller1 får en til tur, og denne turs sum skal være lig scoren
Test data	Kast 1= d1=1, d2=1, score = 10 Kast 2= d1=3, d2=4, score = 7
Expected result	Score = 7
Actual result	Score = 7
Status	Godkendt
Tested by	Oliver Køppen
Data	12/10-2017
Test environment	IntelliJ 2017.2.5 on Windows

Testcase 5 handler om, at spilleren kun kan vinde, hvis personens score er større eller lig med 40, og der slås to ens.

Test case ID	TC05
Summary	Teste, om man vinder ved at slå to ens, når scoren er større eller lig 40
Requirements	
Preconditions	Scorer er lig med eller større end 40 i vinderkastet
Postconditions	Vinderskærmen vises
Test procedure	1. Spiller1 slår to ens 2. Spiller1 vinder
Test data	1. Spiller1 d1=4, d2=4, sum=44 2. Spiller1 har vundet
Expected result	Vinderskærmen vises
Actual result	Vinderskærmen vises
Status	Godkendt
Tested by	Oliver Køppen
Data	12/10-2017
Test environment	IntelliJ 2017.2.5 on Windows

7 | Projektplanlægning

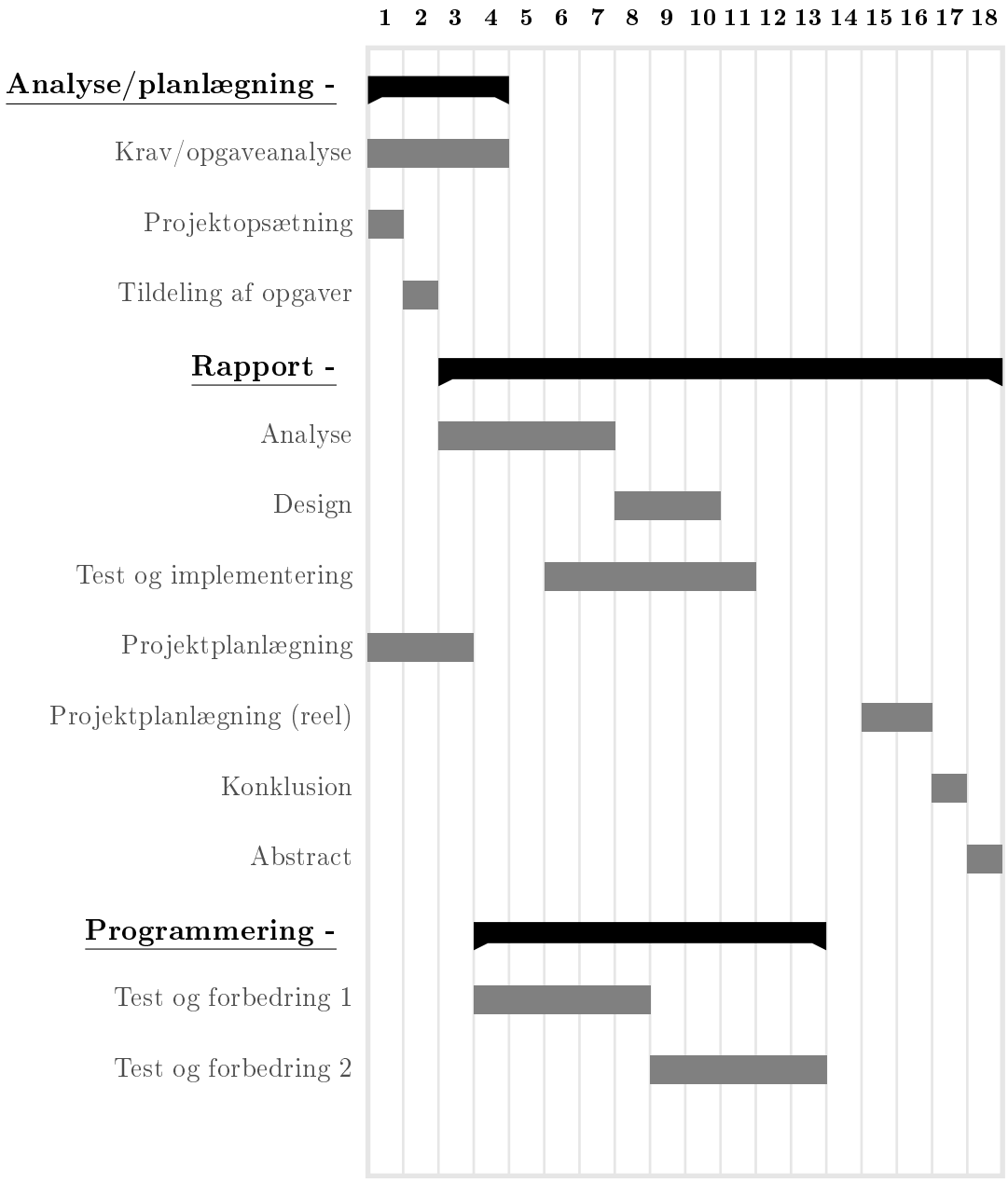
Indledningsvis kom vi ind på, at vi har brugt Asana til projektstyring. På Asana har vi delt projektet op i forskellige *tasks* og uddelegeret opgaverne lige og ladet alle bidrage til alle opgaverne. Det ses i 7.1, at programmet er opddelt i tre spalter; *tasks*, *in progress* og *done*, som dynamisk ændrer sig gennem projektforløbet.



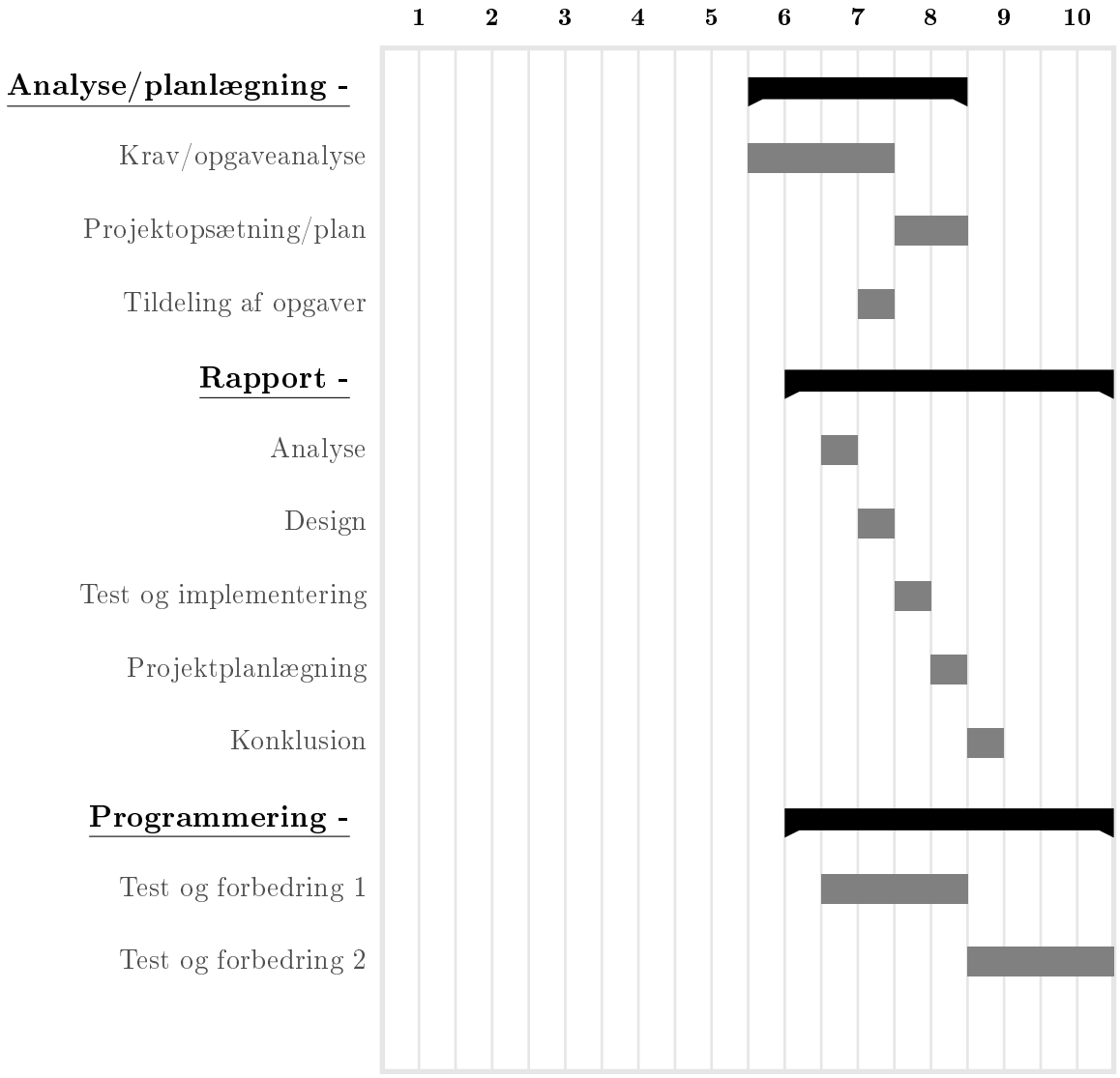
Figur 7.1: Screendump af Asana

Asana understøtter UP, idet man kan oprette x-antal tasks som eksempelvis 'inception' og 'elaboration' og derefter lave x-antal *subtasks*, som kan svare til iterationer.

Diagrammet herunder er inddelt i dage indtil aflevering (18 dage).
Diagrammet viser den planlagte tid, vi lagde fra start af projektet.



Diagrammet herunder er inddelt i dage indtil aflevering (10 dage).
Diagrammet viser den reelle tid der er brugt ved projektet.



8 | Konklusion

Som afslutning på det her projekt kan vi konkludere, at formålet med dette CDIO projekt er opnået. I dette projekt har vi analyseret, designet, implementeret og testet et program At analysere, designe, implementere, og teste et program, der simulere et terningsspil mellem to spillere. Kravene til programmet, herunder størstedelen af de ekstra funktioner stillet i opgaveformuleringen.

Vi gruppens medlemmer, har gennem analysen, udviklingen, samt test af projektet forsøgt at arbejde på en agil måde og dermed følge en arbejdsprocess, der tager udgangspunkt i UP (Unified Process).

Resultatet af vores arbejde er tilfredsstillende. Vi indser, at koden godt kunne have indeholdt flere metoder, istedet for én lang while-/if-statement. Vi forsøgte at lave et flow-diagram, men først efter koden var skrevet, hvilket set i bakspejlet nok ikke var den bedste beslutning. Selvom vi fik opnået et godt stykke arbejde, var der visse områder som vi havde i sinde at forbedre, hvis tidsperioden tilladte dette. I projektformuleringen var der tale om en såkaldt GUI (Graphical User Interface), en brugergrænseflade, som var vedhæftet med i opgaveformuleringen. GUI'en var en del af et matadorspil, hvilket vi kunne anse som en inspirationskilde. Desværre mødte vi lidt besvær under forsøget, nemlig at få pillet noget brugbart ud fra kildekoden. Koden til GUI'en var nemlig en anelse indviklet, og da vi ikke har nogen form for tidligere erfaring med JAVA, havde vi hellere ikke mulighed for at konstruerer en GUI fra bunden. Resultatet af dette var, at vi i gruppen fravalgte, at gøre brug af nogen form for GUI. Selvom vi kunne forestille os, at ved et større projekt, ville vi godt kunne prioritere GUI en anelse mere. Et andet område med plads til forbedring, er planlægningen generelt. På trods af at dette ikke er første gang, hvor vi har arbejdet i grupper, så føler vi at en bedre planlægningen af projektet, ville have gjort en forskel hvad angår tidsfordeling, og kvaliteten af opgaverne.

Alt i alt, er vi tilfredse med resultatet af dette CDIO 1 projekt. Vi mener, at vi har fået et godt førstehåndsindtryk samt en klar ide af, hvordan og hvorledes de kommende CDIO projekter skal gribes an.