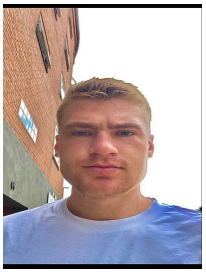
02314 - 62531 - 62532
Indledende Programmering - Udviklingsmetode Til IT Systemer - Versionsstyring og Testmetoder

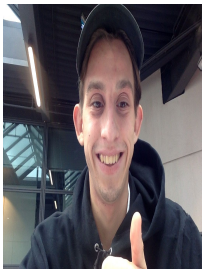
Gruppe 45
02.10.2020
CDIO 1


Alexander Bak Heyde - s193576


Andreas Krone Reichl - s205362


Andreas Borg Kristensen - s205338


Jens Valdemar Lindegaard - s205343


Balder William Stuven - s205359


Kim Michael Randgaard - s205341

Timeregnskab

Navn	**Timer**
Kim Michael Randgaard	15
Balder William Stuven	13
Jens Valdemar Lindegaard	13
Andreas Borg Kristensen	20
Andreas Krone Reichl	15
Alexander Bak Heyde	12

Resumé

Denne rapport omhandler et terningspil, som vi har udviklet ud fra nogle krav stillet af spilfirmaet IOOuterActive. Rapporten starter med en indledning til rapporten, dernæst hvilke krav IOOuterActive har.

Herefter er der en analyse, hvori der er beskrevet hvilke elementer vores projekt skal indeholde, hvilke interessenter vi har med at gøre og om der er eventuelle risici med projektet. Analysen er udarbejdet ud fra UML.

Efter analysen kommer design og implementering, hvori der bliver beskrevet hvordan vi vælger at designe vores spil og hvordan vi får det til at komme til livs.

Til sidst i denne rapport bliver vores testmetoder beskrevet, hvordan vi har planlagt arbejdsperioden og til sidst vores konklusion, hvori vi kan konkludere at vores spil kan leve op til virkeligheden.

Indholdsfortegnelse

Timeregnskab **2**

Resumé **2**

Indholdsfortegnelse **3**

Indledning **4**

Kapitel 1. Krav **5**

Kapitel 2. Analyse **7**

Kapitel 3. Design **11**

Kapitel 4. Implementering **14**

Kapitel 5. Test **18**

Kapitel 6. Projektplanlægning **19**

Kapitel 7. Konklusion **20**

Noter **20**

Bilag **21**

Litteratur- og kildefortegnelse **22**

Indledning

Denne rapport omhandler et spilprojekt hvori vores opgave er at kreerer det. Vi har fået dette projekt af spilfirmaet IOOuterActive. Dette spil er et terningespil hvor formålet er at opnå 40 point. Dette opnås ved at slå med 2 terninger og det antal øjne man får bliver det antal point spilleren modtager. Dette spil spilles af 2 deltagere og vindes ved at opnå de 40 point eller over, hvor i hver spiller har den samme mængde kast.

Vi vil gerne undersøge om vi kan lave et terningespil som kan komme så tæt på virkeligheden og formålet er at det skal bruges til databarerne på DTU og om det kan overholde de teoretiske sandsynligheder. Det spil vi har opbygget er lavet i IntelliJ, som er et IDE, og skrevet i kodesproget java.

Kapitel 1. Krav

Krav til programmet som skal udgøre terningspillet:

Krav	**Uddybning**
System skal kunne køre på windows.	Dette system skal kun kunne bruges ved brug af en Windows computer.
Det skal bestå af to spillere.	Spillet skal kunne håndtere 2 spillere.
Man slår med et raffelbæger som indeholder 2 terninger.	Spillet er sat op til at et raffelbæger skal kunne slå 2 terninger på en gang.
Resultatet af terningernes øjne skal fremstå med det samme.	De to terninger bliver lagt sammen og øjnene der vises er mellem 2-12 øjne.
Summen af terningernes øjne bliver lagt til spilleren som slog med de to terninger.	Efter hvert slag lægges de nye antal øjne lagt sammen med de tidligere slag.
Man vinder, hvis man får 40 point.	Spillet slutter når en af spillerne får 40 akkumulerede points.
Der skal fremvises en test, som viser at raffelbægeret virker korrekt henover 1000 kast.	Vi laver noget statistik ud fra den data som vi samler efter 1000 kast med raffelbægeret. Grunden til vi gør dette er fordi vi gerne vil vise at vores terninger generere "øjnene" tilfældigt.
Systemet skal bestå af følgende pakker: spil og test.	I systemet er der gemt en test-fil, der viser hvordan vi har testet spillet. Foruden er spillet også en del af hele systemet hvor spillere afvikler spillet om og om igen.
Get- og set metoderne i systemet skal være public.	Når metoderne er sat public kan de kaldes ude fra klassen hvor hvis de er sat til private kan de ikke tilgås udefra, som kan ses i vores kode. Get-metoden returnere variablen hvor set metoden tildeler det til variablen.
Systemet skal vise hvis de to terninger viser samme værdi.	Når en spiller slår to værdier, som er ens, vil spillet vise at de to terninger har samme antal øjne.
Der skal gøres brug af tilfældighed generering, når der bliver slået med terningerne.	Systemet er sat op med en tilfældighedsgenerator (*Math.random*) så hvert udfald af tegningerne er tilfældigt genereret.
Koden skal kunne inspiceres til senere brug.	Vi arbejder med et Java, hvor vi gerne vil kunne gå tilbage og ændre i vores kode, hvis vi gerne vil implementere flere funktioner til spillet og gøre det mere avanceret.

I testen skal der optælles hvor hyppigt terningerne bliver ens.	Vi har lavet tre test på baggrund af vores tilfældighedsgenerator af 1000 slag. Derfra har vi lavet afbildning af resultaterne og lavet tre separate diagrammer som kan findes i vores bilag.
Testene der bliver udført skal kontrolleres, at de stemmer overens med de teoretiske sandsynligheder.	På baggrund af teoretisk statistik om sandsynlighed, vil vi sammenligne vores udfald fra 1000 terningekast, og se om det stemmer overens med vores teoretiske forventninger.

Kravspecifikation:

Væsentlige detaljer, som projektet skal indeholde

- Konceptet for projektet er at lave et terningspil hvor to spillere kan spille, hvor vinderen findes ved den som får 40 point eller derover først.
- Målsætningen med projektet er at spillet skal fungere på DTU's databarer.
- Den defineret målgruppe er for personer som har adgang til DTU's databarer og herved kan gøre brug af spilprojektet ved at spille det.
- Tidsplanen for projektet er 14 dage.
- Projektets ydeevne skal være at kunne spille terningspillet hertil skal der også være en mulighed for at projektet kører en test henover 1000 kast.

Flow af spillet

Spillet skal spilles af 2 spiller.
Efter hver runde bliver antal øjne lagt sammen.

1. Spiller 1 slår
2. Får et antal øjne
3. Spiller 2 slår
4. Får et antal øjne
5. 1-4 gentages indtil en Spiller opnår 40 point
6. Første spiller der opnår 40 point, vinder

Data og databasen

Dataene som der bliver gjort brug af i runderne af terningspillet vil blive midlertidigt gemt i form af hvem der vandt de foregående runder.

Der gøres brug af en database som skal undersøge hen over 1000 kast at vores projekt stemmer overens med de teoretiske sandsynligheder som forbindes med en terning. Den skal også vise hvis terningerne har samme værdi.

Administrationspanel

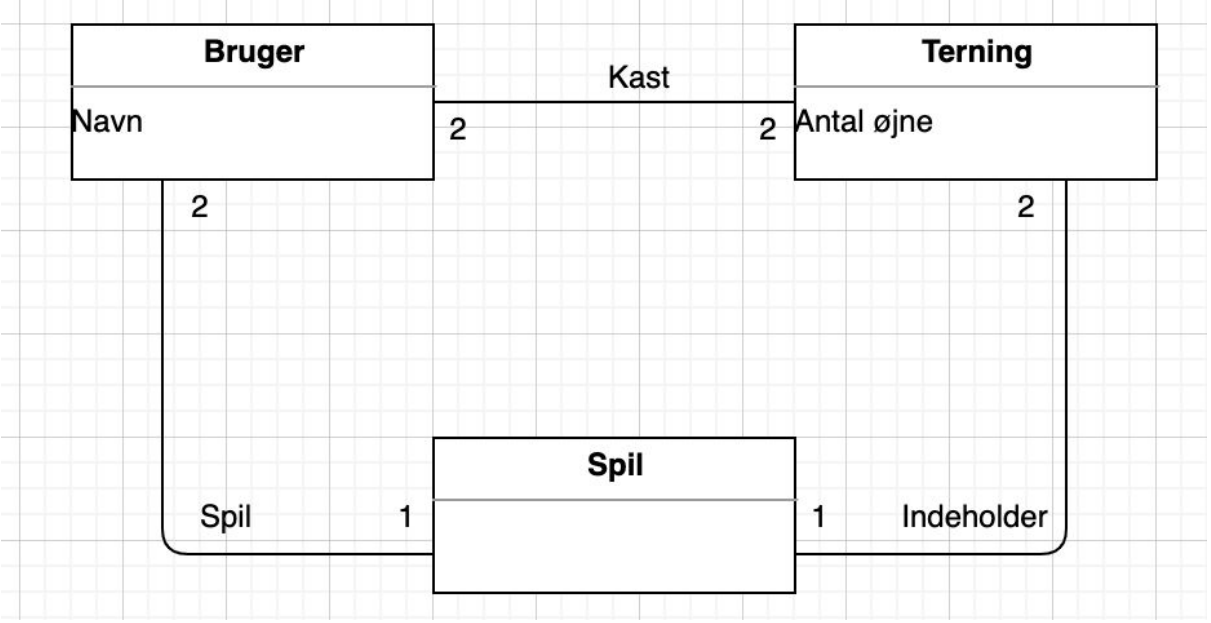
Her skal det være tilgængeligt for kunden at se de forskellige versioneringer af projektet og inspicere koden og udviklingen i projektet. Kunden skal også have adgang til de forskellige grene og der skal være ændringer i udviklings grenen som ikke er kørt og testet.

Kapitel 2. Analyse

Use cases
Ud fra hvilke krav der er blevet stillet har vi udarbejdet disse use cases.

Use cases	**Uddybning**
Skriv navn	Man skal skrive sit navn
Start Spil	Når man vil starte spillet
Slå terninger	Man slår med terningerne
Spil igen?	Mulighed for at spille igen
Slut spil	Mulighed for at afslutte spillet

Domænemodel
Vi har bygget vores domænemodel ud fra hvor mange spillere der kan deltage og hvor mange terningerne der er i spil.
På figuren (*Figur 1*) kan der ses hvordan vores domæne hænger sammen.



Figur 1. Dette er vores domænemodel.

Hvilke aktører har vi med at gøre?

- Udgiver

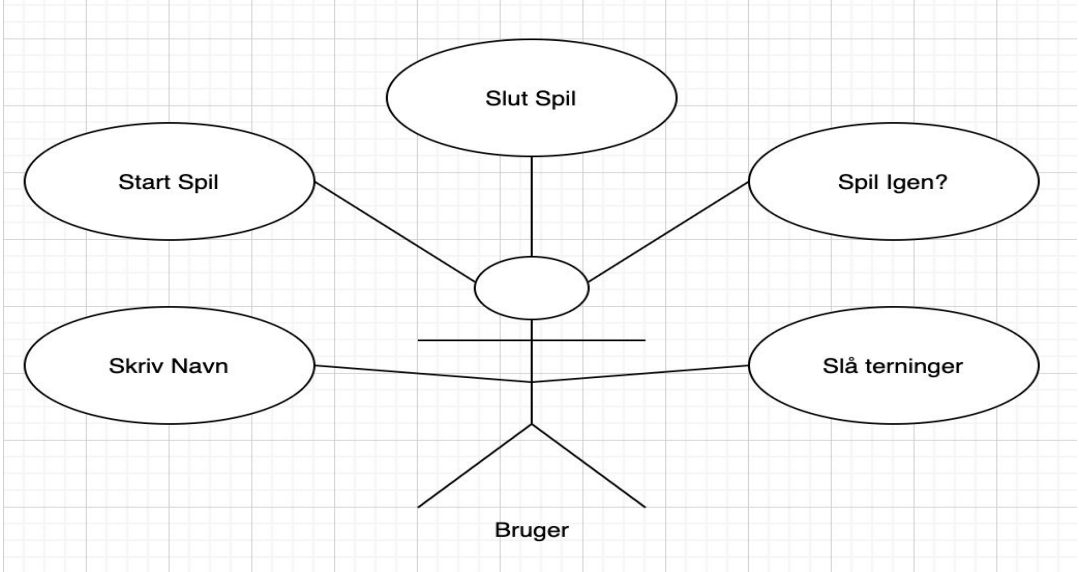
Vi har med en udgiver/arbejdsgiver at gøre med. Dette projekt er vi blevet tildelt af IOOuterActive, og det er dermed dem, der har sat krav til projektet, og dem der har en bestemt vision, for hvad projektet skal indeholde. Det er også dem, som vælger hvor det skal distribueres.

- Programmør

Dette er os, som sidder og koder for projektet. Det er os som står for at projektets kodning er i orden og fungerer, og eventuelt opdaterer projektet hvis der er brug for dette. Vi sidder også og tester projektet løbende.

- Bruger

Brugerne er dem som kommer til at benytte projektet. De har også mulighed for at hjælpe projektet ved at komme med feedback til diverse ting som enten kan tilføjes, ændres eller fjernes.
Brugerne er de primære aktører, fordi det er dem som skal benytte vores projekt.



Figur 2. Dette diagram viser vores use cases ud fra vores bruger aktør.

Hvilke interessenter har vi?

- Udgiver

IOOuterActive har en stor indflydelse på vores projekt, fordi det er, som skrevet tidligere, deres vision og idéer vi går ud fra. Deres medvirkning er til gengæld ikke stor, da de ikke aktivt er med til opbyggelsen og udførelsen af projektet.

- Programmør

Vi, programmører, har både stor indflydelse og medvirkning, fordi vi er essentielle for at projektet bliver lavet, og vi har mulighed for selv, at udvikle og tilføje features til projektet.

- Bruger

Brugeren har ikke ingen medvirkning for gennemførelsen af vores projekt, men de har en indflydelse på det. De har muligheden for at ændre og/eller forbedre projektet, ved bare at benytte det og komme med feedback.

Hvilke krav er gældende for vores interessenter og hvem er de vigtigste?
Krav

- Udgiverens krav

IOOuterActive, som er vores udgiver, har nogle krav i forhold til hvad vores projekt skal indeholde, og hvor det skal lanceres henne.
Det skal kunne bruges på Windows, det skal indeholde 2 spillere, det skal indeholde 2 terninger, man skal kunne se resultatet med det samme og man vinder spillet ved at opnå 40 point. Man får point ved at lægge summen af øjnene på terningerne sammen.

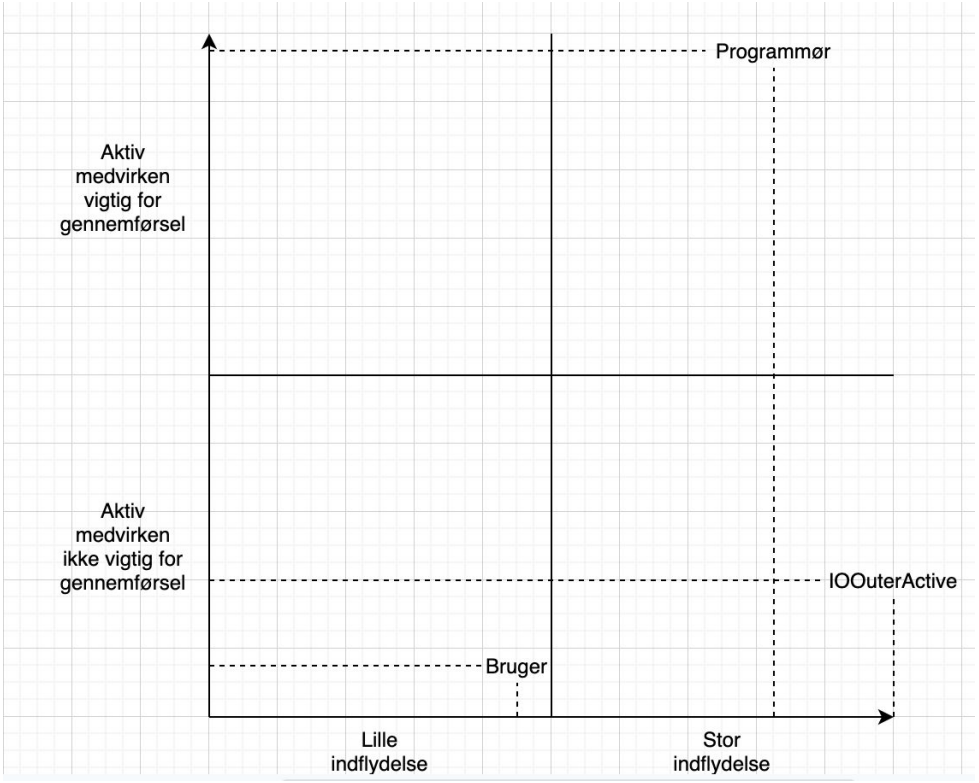
- Programmørens krav

Vi har stillet krav til at spillet skal være "fair", som i vores optik er at begge brugere skal have mulighed for at kaste med terningerne hver runde, da vi ellers mener at den person der starter har større chance for at vinde. Der skal også være interaktion mellem brugere og spillet, så spillet ikke bare kører af sig selv.

- Bruger

Brugerne har ingen krav til spillets udgivelse.

Vigtigste interessenter
De vigtigste interessenter i dette projekt er udgiveren og programmørens. Brugerne er ikke særlig vigtige i forhold til at udgive spillet, men deres vigtighed kommer efter udgivelsen, fordi deres feedback er nyttig, for at vide om projektet er en success.



Figur 3. Dette diagram visualiserer hvor vigtige vores interessenter er.

Hvilke risici er der/kan der opstå

- Systemnedbrud

Hvis maskinen projektet kører på slukker på uventet vis eller projektet selv bryder sammen. Dette burde dog ikke forekomme.

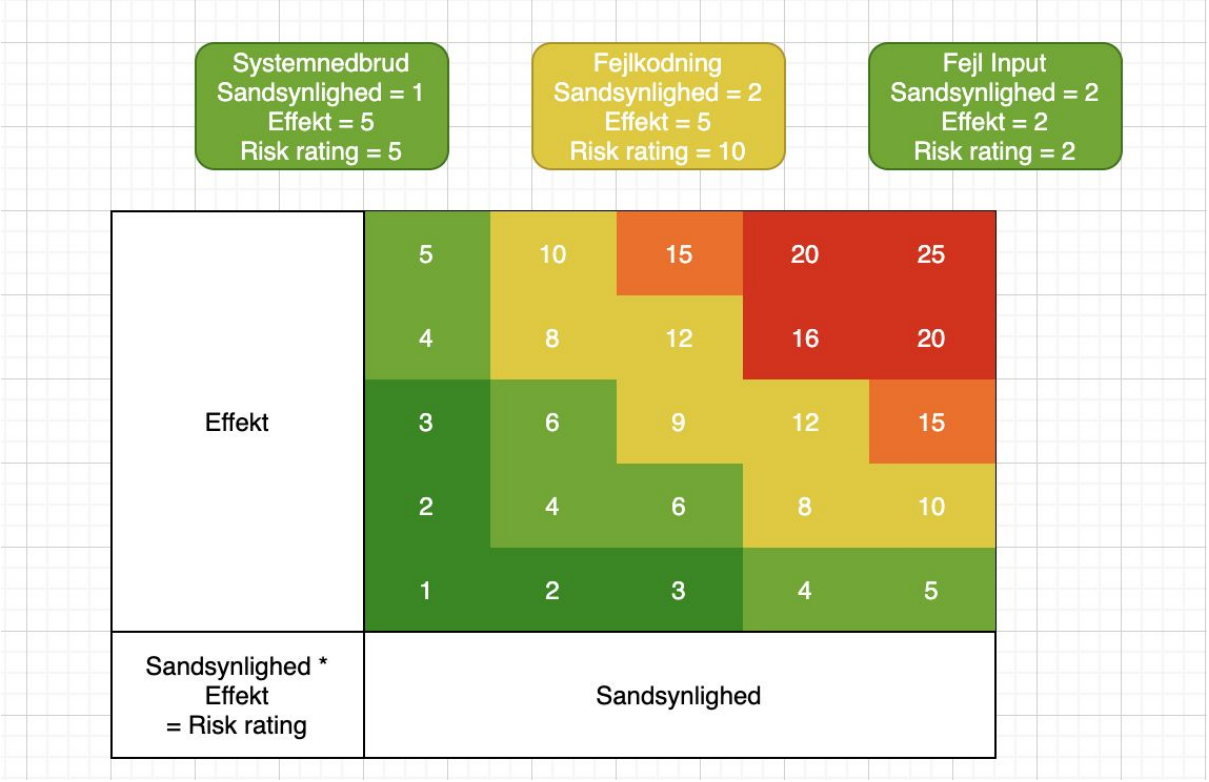
- Fejlkodning

Hvis vi ikke koder programmet til at opfylde kravene og/eller at det slet ikke fungerer på den måde vi har håbet. Dette kan forekomme, men er løsbart. Dette problem er løsbart ved at gennemgå koden for fejl flere gang.

- Fejl Input

Brugeren kan komme til at taste forkert og dermed vil programmet ikke korrespondere "rigtigt" til det forventede. Dette kan forekomme og vil have indflydelse på oplevelsen af spillet. Dette problem kan vi forebygge bedst muligt ved at give klare instrukser i projektet.

Der kan ses på figuren herunder (*Figur 4)* hvor alvorlige de forskellige risici er. I vores projekt er vi heldige ikke at have nogle alvorlige (15+ risk rating) risici.



Figur 4. Dette er en probability/impact matrix som viser hvor sandsynligt, hvor stor en effekt vores risici har og hvor alvorlige de er.

Kapitel 3. Design

Løsningsdesign

- Hvad er vores konkrete løsning?

Løsningen til spilprojektet, er opbygningen af et terningspil, hvor to spillere indgår i processen, der skal henholdsvis kastes to terninger som går på tur. Her handler det for de to spillere om at slå 40 point(først), hvor terningernes øjne svarer til antal point spilleren modtager, hvis en spiller har slået 40 point eller derover skal personen slå et par for at vinde runden. Her er undtagelsen dog par 1, som vil nulstille spillerens point.

Spilprojektet er blevet konstrueret i java, her bliver der vist i terminalen henholdsvis spillernes slag og deres midlertidige point. Der bliver også tydeliggjort hvem vinderen af spillet er.

- Hvordan løses problemet i problemformuleringen?

Problemet i vores problemformulering om hvordan et terningspil bliver konstrueret, så det overholder at der kun er to spillere, hvor den der slår 40 point eller derover vinder. Bliver løst ved at opstille nogle parametre for vores java program, således at det kun er tilladt at være to spillere ad gangen. Der bliver også opstillet at den der slår 40 point først vinder, og hvis begge spillere slår 40 point, vil det være den som slår to ens uden at den anden spiller ikke slår par som vinder.

Enkeltløsning

- Hvad er løsningen?

Løsningen er et terningspil, hvor to spillere spiller om at få 40 point.

- Hvilke funktioner har løsningen?

Løsningen har til funktion at skulle kaste to terninger i et raffelbæger, her skal turen gå fra den ene spiller til den anden, indtil der er en som har slået et sammenlagt pointantal som giver 40. Her vil programmet så vise antallet af runder de enkelte spillere har vundet, så vil den spille terningspillet færdigt, indtil antallet af runder man valgte i begyndelsen af spillet er opfyldt.

- Hvordan elementerne i løsningen interagerer med hinanden?

Elementerne som terningerne, antallet af runder og det samlet antal point den enkelte spiller har interagerer med hinanden, for at finde en sammenlagt vinder for terningspillet. Terningerne bestemmer hvilke point spillerne får og dermed spiller de en direkte rolle i hvem som vinder runderne, det er på denne måde at elementerne i løsningen interagerer med hinanden.

- Hvordan lever løsningen op til de enkelte krav i kravspecifikationen?

Der bliver brugt en klassisk metode til brug af løsningsdesignet for vores spilprojekt. Den klassiske metode er hvor vi først har fået defineret krav til spil projektet og derefter udviklede

vi en løsning som kunne afdække kravene og kravspecifikationen, herved sikrer vi os, at løsningen lever op til de enkelte krav i kravspecifikationen.

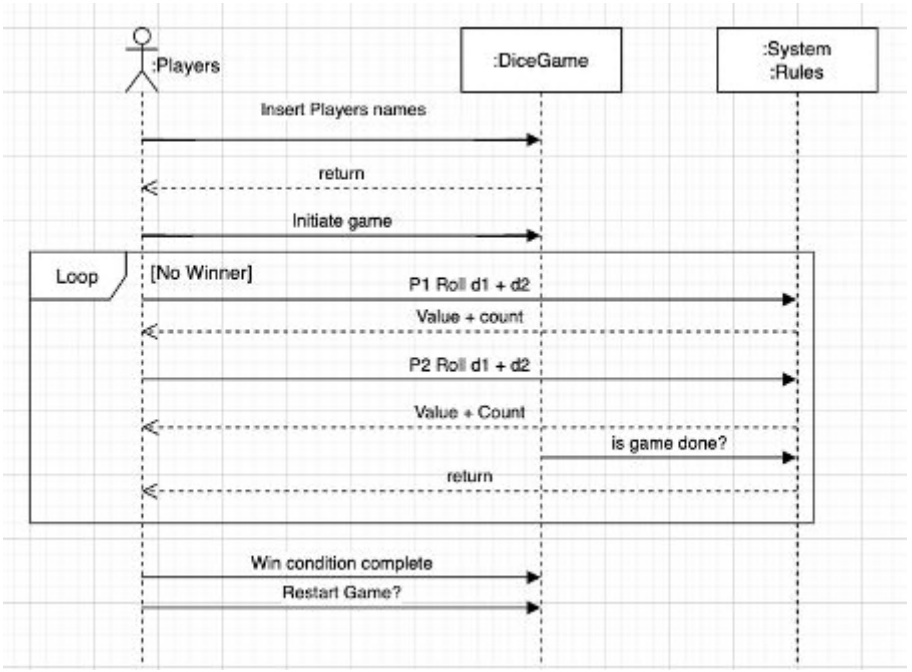
Flow af vores spil

1. Åbn spillet.
2. Spillet forklarer reglerne til begge spillere inden spillets start.
3. Regel 1: Når du rammer 40 point eller derover, skal du kaste et par for at vinde spillet.
4. Regel 2: Hvis du kaster et par, får du et ekstra kast.
5. Regel 3: Hvis du kaster terningerne og slår par 1, mister du alle dine point.
6. Regel 4: Hvis du kaster par 6 to gange i streg, vinder man spillet.
7. Spiller 1 indtaster deres navn.
8. Spiller 2 indtaster deres navn.
9. Derefter bliver der indtastet "Start" af en af spillerne, dernæst starter spillet.
10. Nu startes spillet.
11. Dernæst trykker spiller 1 "k" for at kaste begge terningerne på samme tid.
12. Spiller 1 får vist hvad hver terning har kastet og derefter en sum af terning 1 og terning 2.
13. Herefter får spiller 2 muligheden for at trykke "k" så spillet kaster terningerne.
14. Spiller 2 får vist hvad hver terning har kastet og derefter en sum af terning 1 og terning 2.
15. Fra andet kast af spiller 1 og 2, bliver summen af terningerne ved det nuværende kast, tilføjet til den akkumuleret værdi.
16. Nu bliver bruger flows fra 11 - 15 gentaget indtil enten spiller 1 eller spiller 2 har opnået 40 point.
17. Derefter træder regel 1 i spil, og spiller 1 og spiller 2 skiftes mellem at slå.
18. Hvis spillerne over 40 point, slår par 1 mister de alle deres point og starter forfra.
19. Den spiller som har over 40 point og slår et par, vinder.
20. Den vindende spiller får 1 point i banken
21. Så spørger spillet om spiller 1 og 2 har lyst til at spille endnu en runde.
22. Ved at de trykker "y", starter ny runde fra punkt 11.
23. Og dernæst begynder spillet så man spiller bedst ud af tre
24. Ved at de trykker "n" lukkes spillet.

Sekvensdiagram

Først og fremmest starter spillet med at spørge spillerne om at indtaste deres navne.
Når det er registret i spillet, er spillet nu klar til at starte. Spiller starter spillet hvor der efterfølgende kører en løkke som er selve terningeslaget. Det er back-end systemet der registrere turens point og akkumulerede point for spillerne hvilket bliver returneret til spillerne i terminalen. "Is game done?" ser på hvilke krav og regler vi har sat op for spillet, som bliver checket hver gang begge spilleres tur er overstået.

Når en spiller har fået de fyrre points og slår et par eller to par 6 i træk, opnår de vinder krav. Og spillet er nu klar til at køre igen.



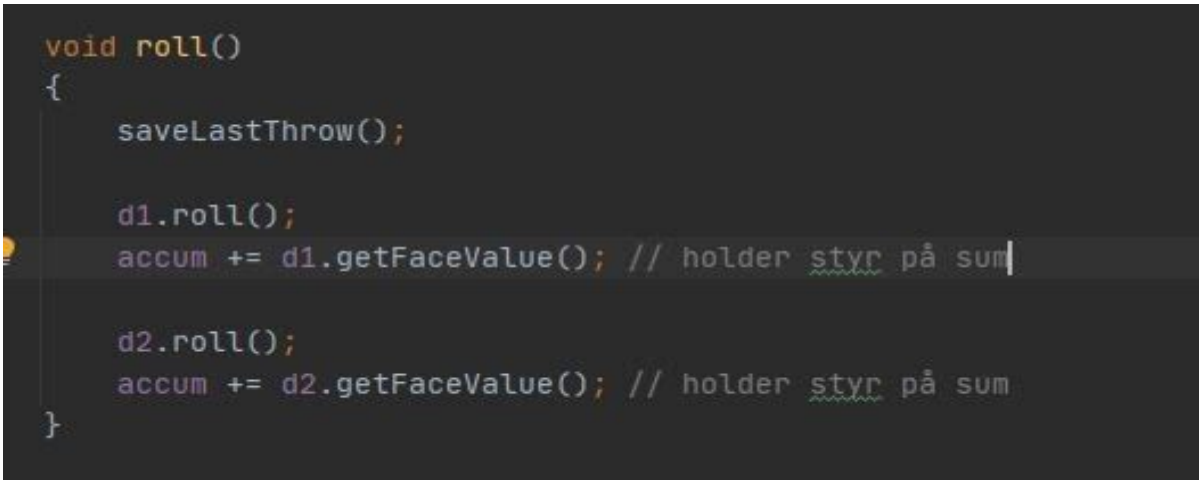
Figur. 5 Dette sekvens diagram viser en tidslinje fra start til slut i vores spil og hvilke interaktioner som spiller går igennem med spillet og systemet.

Kapitel 4. Implementering

Implementering af koden

```
public void roll()
{
    faceValue = (int)(Math.random()* MAX +1); // Ruller et tilfældigt tal mellem 1 og 6
}
```

Vi benytter os af Math.random funktionen fra intellij til at generere et tilfældigt tal mellem 1 og 6, hvor 6 er vores MAX værdi. Vi definerer udfaldet som faceValue, som vi kan benytte senere i koden.

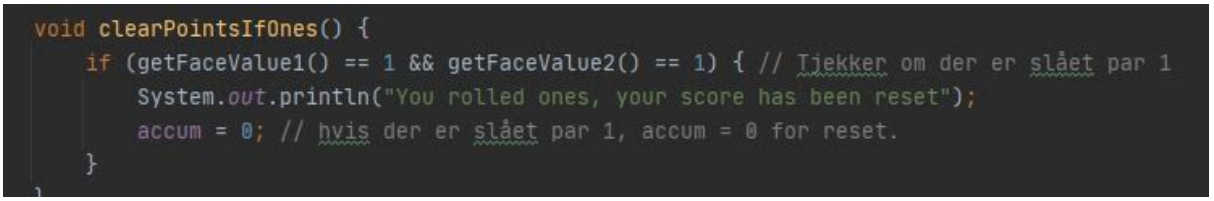


```
void roll()
{
    saveLastThrow();

    d1.roll();
    accum += d1.getFaceValue(); // holder styr på sum

    d2.roll();
    accum += d2.getFaceValue(); // holder styr på sum
}
```

I denne kode gør vi brug af faceValue fra forrige billede. Vi har to terninger, som vi kaster samtidigt ved et roll, hvortil d1 og d2 er de to terninger. Accum er den samlede sum, hvor vi tilsætter kastets faceValue til. Grunden til, at vi har valgt at bruge to terninger, er for at interagere bedre med brugeren, så det relaterer mere til et virkeligt spil.



```
void clearPointsIfOnes() {
    if (getFaceValue1() == 1 && getFaceValue2() == 1) { // Tjekker om der er slået par 1
        System.out.println("You rolled ones, your score has been reset");
        accum = 0; // hvis der er slået par 1, accum = 0 for reset.
    }
}
```

I denne kode tjekker vi, at hvis faceValue på terning 1 = 1 og faceValue på terning 2 er = 1, skal accum sættes = 0, da man taber alle point ved dette slag.

```
boolean extraTurn() {
    return getFaceValue1() == getFaceValue2(); //tjekker om der er slået par, for at få ekstra slag.
    }
```

En anden regel er, at hvis der bliver slået par, skal personen have et ekstra slag. Vi tjekker ved hjælp af ExtraTurn, at hvis FaceValue1 = FaceValue2, får vi et ekstra slag.

```
boolean isCurrentAndLastThrowSixes() {
        return lastThrow == 12 && (getFaceValue1() + getFaceValue2()) == 12; //holder styr på om der er slået 2x6 to gange i streg
}
```

Her holder vi styr på, hvorvidt vi har slået dobbelt 6, to gange i streg. Vi har gemt vores sidste slag i lastThrow, og siger, at den skal return lastThrow = 12 og faceValue1+faceValue2 = 12.

```
boolean isGameDone() {
    boolean isFinalStage = (getAccum() - (getFaceValue1() +getFaceValue2())) >= 40; // tjekker om spilleren er kommet over 40 point
    return (isFinalStage && getFaceValue1() == getFaceValue2()) // tjekker spiller har over 40 point ved par, for at vinde spil
            || isCurrentAndLastThrowSixes(); // Tjekker om der er slået 2x6 for at vinde spil
}
```

I denne boolean isFinalStage tjekker vi, om spilleren er kommet over 40 point. Hvis spilleren er kommet over 40 point, så siger vi, at den skal returnere isFinalStage og hvorvidt vi har slået par, for at vinde spillet. Ellers tjekker vi for om der er slået dobbelt 6, 2 gange i streg.

```
private static void doTurn()
{
    do {

        // spørges om de er klar til at slå, ved k kaster de
        System.out.println(currentPlayer.getNavn() + " Press 'K' if you're ready to throw");
        sc.next();

        currentPlayer.roll(); // spilleren kaster terningerne

        //Navn, terning 1 og 2's værdi og den akkumulerede printes ud.
        System.out.println(currentPlayer.getNavn() + " rolls: (" + currentPlayer.getFaceValue1() +
                "," + currentPlayer.getFaceValue2() + ") ->" + currentPlayer.getAccum());

        currentPlayer.clearPointsIfOnes(); // hvis spilleren slår par 1, bliver den akkumulerede sat til 0.

        // løkken køres så længe spilleren har et ekstra slag og spillet ikke er færdigt
    } while (currentPlayer.extraTurn() &&! currentPlayer.isGameDone());
}
```

Her ser vi programmet for at køre en runde i vores spil.
Vi starter med at bede Currentplayer om at trykke k, hvis han er klar til at slå. Herefter kaster han terningerne, med funktionen roll. Vi udskriver så Spillerens navn, hans slag og hans samlede antal point. Vi kalder funktionen clearPointsIfOnes, for at tjekke hvorvidt han har slået par 1, og dermed skal have nulstillet sine point.
Dette kører i et do-while loop, hvor det skal fortsætte så længe den pågældende spiller har fået et ekstra slag, ved brug af funktionen extraTurn, og så længe spilleren ikke har opfyldt kravene for isGameDone.

```
private static void playGame()
{
    int round = 1;

    //loop indtil hverken spiller 1 eller 2 har opnået spillet er færdigt
    while ((!p1.isGameDone()) &&! (p2.isGameDone()))
    {

        System.out.println("Round: " + round);
        round++; // printer runde nummer ud og plusser med en for hver runde.

        currentPlayer = p1;
        doTurn(); // kører do loop på player 1
        currentPlayer = p2;
        doTurn(); // kører do loop på player 2

        // printer current score ud for player 1 og player 2
        System.out.println("Current score is: " +  p1.getNavn() + ": " + p1.getAccum()
                + " and " +  p2.getNavn() + ": " + p2.getAccum());
        System.out.println();
    }
```

Her ser vi koden for vores playGame. Vi starter med at printe antallet af runder spillet ud, og hvergang en runde er færdig spillet, ligger vi en til.
Vi sætter så currentPlayer = p1, og kører vores do-while loop på pågældende spiller.
Herefter sætter vi currentPlayer = p2, og kører samme loop. Når begge spillere har spillet deres runde, printer vi current score ud for begge spillere.
Dette forsættes i vores while loop, så længe hverken spiller 1 eller spiller 2, har opfyldt kravene for isGameDone.

```
    // Spillet er færdig og der spørges om de vil spille igen
private static void newGame() {
    inputPlayerNames();
    String another = "y";

    while (another.equalsIgnoreCase( anotherString: "y"))
    {
        playGame(); //Kalder playGame hvis de vil spille igen
        System.out.println();
        System.out.print("would you like to play again? (y/n)");
        another = sc.next();


        p1.newGame(); // spiller 1 reset for nyt spil
        p2.newGame(); // spiller 2 reset for nyt spil
    }
}
```

Slutteligt viser vi implementeringen for newGame.

Har spiller 1 eller spiller 2 vundet, bliver de spurgt hvorvidt de vil spille igen. Trykkes der y, starter spillet forfra og begge spillere bliver reset til newGame, hvor deres sum og lastThrow bliver sat til 0. De bliver så dirigeret op til vores playGame igen, hvor spillet starter igen. Trykkes der n, afsluttes programmet og spillet stoppes.

Implementering af løsning

Firmaet vil implementere vores spilprojekt på databarer på DTU, her har vi valgt, at bruge engelsk som sproget spillet bliver udgivet i, fordi det vil ramme en større målgruppe, da der også er folk på DTU som kun snakker engelsk.

Der er størst mulig succes for projektet har vi derfor valgt at det kan implementeres til så bred en målgruppe som muligt.

Kapitel 5. Test

Test af løsning

- Har løsningen den effekt vi vil opnå?

Den effekt vi gerne vil have, at vores løsning skal opnå, er en visning at vores tilfældighedsgenerator i vores spilprojekt, stemmer overens med de teoretiske sandsynligheder som der ville opstå under et terningspil. Det har vi vist ved at lave tre simuleringer af programmet i en testfil hvor vi har kørt selve udfaldet af vores tilfældighedsgenerator 1000 gange for hvert test. Vi har kørt programmet af tre omgange og opstillet det i en tabel som vist nedenfor.

Øjne	Sim 1	Sim 2	Sim 3	gennemsnitlig fordeling (166.66)
1	153	158	175	162
2	163	168	152	161
3	191	170	159	173.33
4	156	153	170	159.66
5	171	185	184	180
6	166	166	160	164

Her har vi en tabeloversigt over hvor mange gange der er blevet slået hhv. 1-6 ud fra de tre simulationer. Det vi kan aflæse fra tabellen er der er en tilfældighed ved vores program. Kigger vi eksempelvis på hvor mange gange der er blevet slået en 3'er så kan vi se at i første simulering er der blevet slået 191, i den anden simulering 170 gange og i den tredje simulering er der blevet slået 159 gange. Vi kan herfra blot ud fra tallene se at vores program er bygget op med en tilpas stor tilfældighed som gør at spillet vil have forskelligt udfald hver gang vi spiller spillet.

Se bilag 1 for diagram for de 3 gennemgange.

Kapitel 6. Projektplanlægning

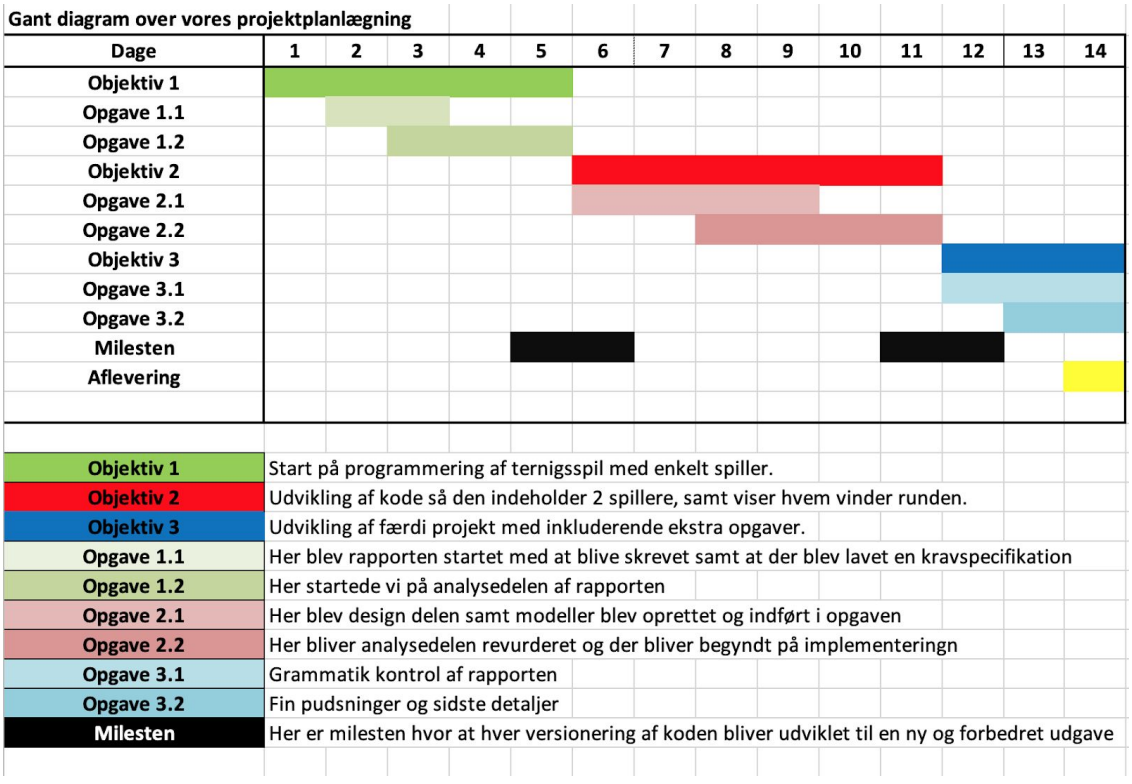
Beskrivelse af det planlagte- og gennemførte processer i projektforløbet:
Tidsplanlægningen kan yderligere inspiceres i *Figur 6,* i denne model er der vist hvor lang tid der er brugt på enkelte dele i vores projekt.

- Planlagte

Vi havde planlagt vores projektforløb således at da koden blev skrevet blev der også udarbejdet enkelte kapitler i vores projektrapport, dette skabte at der var god sammenspil mellem hvad man faktisk fik produceret i fohold til et reelt program og hvordan vi var kommet frem til de enkelte dele ved f.eks brug af domænemodellen eller kravspecifikationerne.

- Gennemførte

Alle planlagte dele af vores projektforløb blev gennemført givet af god tidsdisponering, her blev der også taget højde for ekstra opgaverne som blev stillet til projektet. Dette blev placeret i den sidste del af vores tidsramme da det var den sidste del som skulle implementeres.



Figur 6. Dette er et gantt diagram som beskriver vores projektplanlægning og hvordan vi har tilrettelagt vores tid i forhold til projektet.

Kapitel 7. Konklusion

Vi kan konkluderer at vores spil lever op til vores forventninger. Det opfylder IOOuterActive krav, vores egne krav og bygget op ud fra vores definition af fair. Vores test viser at vores tilfældighedsgenerator virker og at vi kommer tæt på hvordan et terningespil fungerer i virkeligheden.

Noter

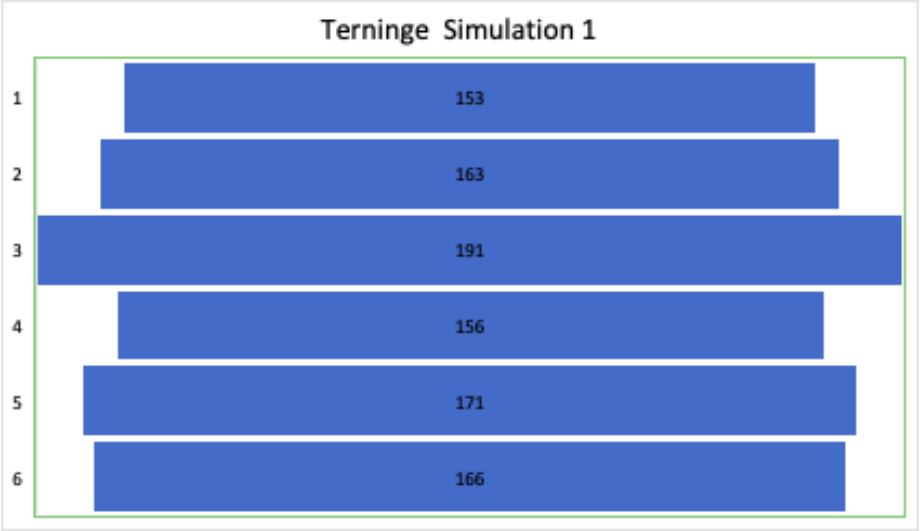
Som alle andre systemer, kan vores også optimeres, som vi efterfølgende har fundet ud af. Eksempelvis kunne man implementere, at spillerne kunne vælge i starten om de vil spille almindeligt spil, først til 40 point, eller den avanceret udgave med de ekstra regler.
Vi så gerne at vores system også havde tildelt en GUI som gerne skulle kunne afbilde hvad der sker visuelt.

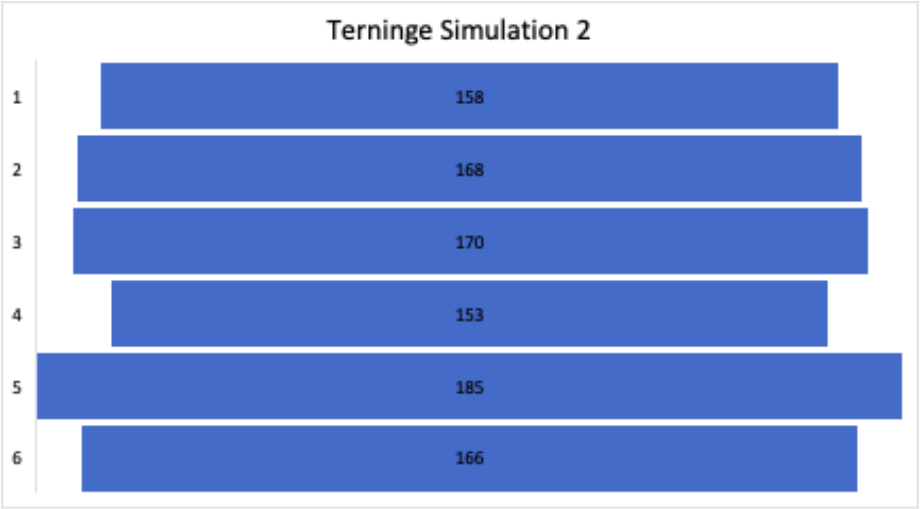
Som en revurdering af hvad vi kunne lave om til næste gang, har vi udarbejdet følgende punkter:

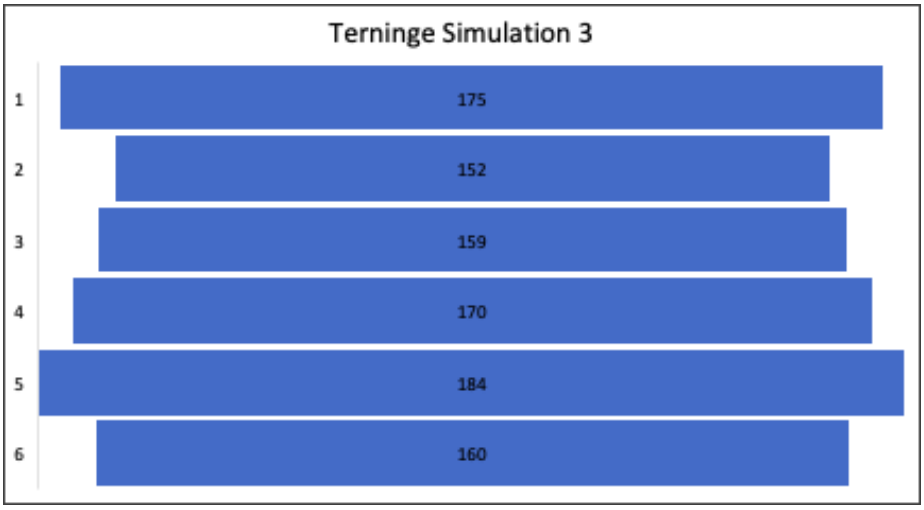
- Arbejdsfordeling/arbejdsindsats for hver enkelt rapport og medlem.
- Gruppekontrakt
- Forventningsafstemning

Bilag

Bilag 1 - Fordeling af 1000 terningeslag.







Litteratur- og kildefortegnelse

Craig, Larman. (2019). *Applying UML And Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3. udg.). Pearson Education (Us). (Originalværk udgivet 2004)

John Lewis, JL. (2017). *Java Software Solutions, Global Edition* (9. udg.). Pearson Education Limited.

Samuel Brüning Larsen, SBL. (2018). *Projekter og rapporter på tekniske uddannelser*. Hans Reitzel.