62531, 62532, 02312

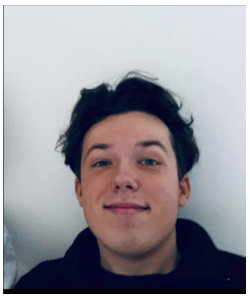
Udviklingsmetoder til IT-systemer, Versionsstyring og Testmetoder, og Indledende Programmering

---

CDIO 2

01. Oktober 2021

Sebastian N. B. E.



Christian H.

Valdemar N.

Sebastian G.

Lasse G-J.

Resumé

- Vi skal, som udviklere for spillefirmaet IOOuterActive, analysere, designe, implementere og teste et terningespil, som opfylder kundens krav og vision.

Timeregnskab

- Sebastian G: 10 time 30 min
- Sebastian E: 7 timer 30 min
- Christian: 7 timer 30 min
- Valdemar: 7 timer
- Lasse: 9 timer

Indholdsfortegnelse

Indledning

I denne rapport vil vi komme ind på planlægning og opbygningen af vores kode og projektet som helhed.

Link til github repo :https://github.com/ValdemarNielsen/CDIO2

Projektplanlægning

Udarbejd et terningespil efter kundens vision som skal være klar d. 29/10/2021. Udvikling af projektet foregår med løbende opdateringer på Github og en tidlig start, hvor vi også har udarbejdet modeller til at gøre projektet overskueligt og forståeligt for alle i gruppen.

Analyse

Krav:

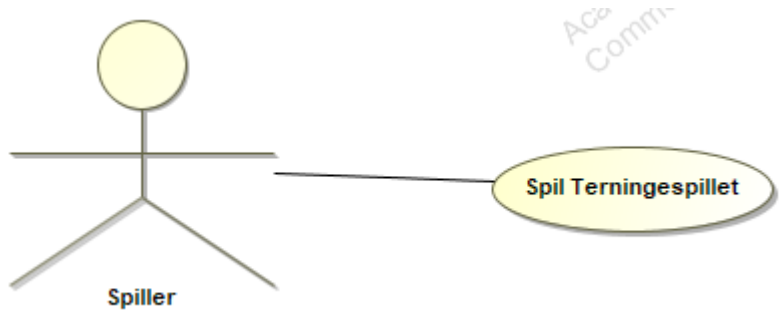
Vi skal ligesom i CDIO 1 lave et spil der spilles af to spillere. Dette spil spilles med to seks sidede terninger. Hver spiller skiftes til at slå med terningerne og lander derefter på et felt, disse felter påvirker spillerens pengebeholdning, og om man må slå igen. Vi vil komme mere ind på hvordan spilles spilles i vores use case beskrivelse. Et krav i forhold til versionering er at kunde ønsker at kunne følge processen, derfor er de interesserede i at se en historie af de commits der er lavede osv. Det er her vigtigt at committe ret ofte da det derfor bliver nemmere at følge med i hvad der sker. Kunden har derfor adspurgt at der skal commites mindst en gang i timen eller hver gang en delopgave er løst. Desuden skal der skrives hvad der er blevet lavet, dette gør det nemmere for alle at følge med i hvad der er blevet lavet. Kunden ønsker også at dette program er godt og virker derfor vil kunden se nogle tests. Kunden foreslår Junit. Denne ene test skal vise at det er meget usandsynligt en spillers bankkonto går i minus. Desuden skal vi også lave andre test der viser at programmet virker som det skal. Kunden ønsker desuden at vide hvilke specifikationer der er krævede for at spille spillet. Kunden har et krav til koden der hedder at vi skal vise vi kan bruge klasser og relationer til at løse denne opgave.

Desuden er der også en lang række af løsere krav. Disse krav er ses for eksempel til koden, hvor der kræves vi laver ting i en bestemt rækkefølge og andre krav, hvor vi bare skal argumenter hvis vi ikke vil overholde dem til punkt og prikke.

Use case:

For at finde ud af, hvor vi skal starte med dette program vil vi lave en use case, der beskriver programmets forløb. I dette program er der kun en use case nemlig: at spille terningespillet. Da der kun er en use case er dette selvfølgelig også main use case. Da der er tale om et repetitivt spil i ture vil use casen beskrive hvordan en tur foregår, hvordan spillet starter og hvordan der findes en vinder. Der er kun en aktør i denne use case.

Use case diagram:



Use case navn: Spil terningespillet

Scope: Terningspillet

Level: User goal

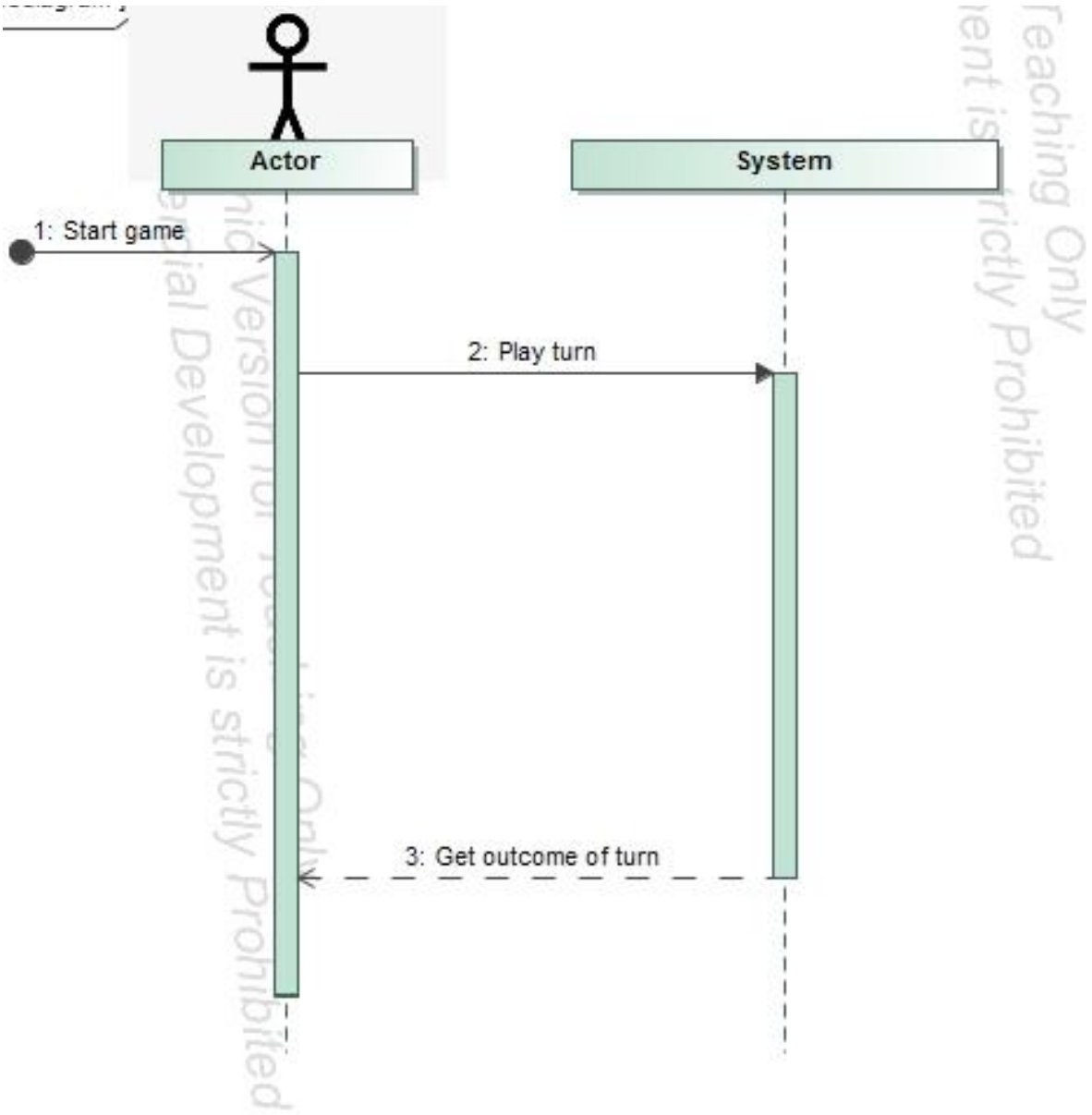
Primær aktør: Spilleren

1. Spillet starter
2. Begge spillere starter med 1000 points på deres konto
3. Spiller 1 slår med 2 terninger med 6 sider
4. Spiller bliver tildelt et antal points ud fra summen af terningerne
5. Hvis spilleren slår og får en sum af 10, må spilleren slå igen
6. Hvis spilleren får 3000 points eller over vinder spilleren

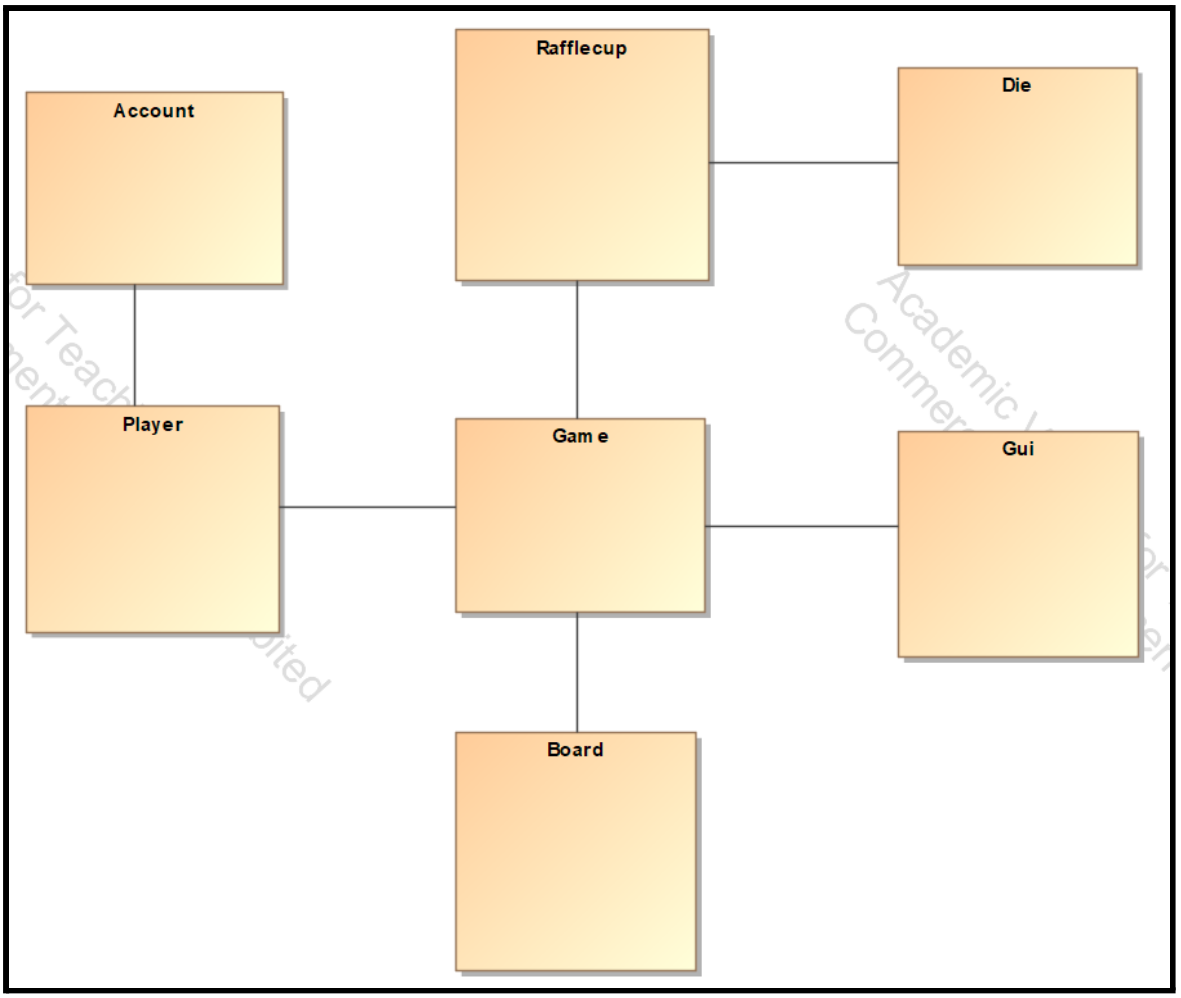
7. Spillet er nu slut

Systemsekvensdiagram:

Herunder ses et systemsekvensdiagram,Dette diagram beskriver hvad der sker nå aktøren spiller en tur i spillet. Dette diagram er meget simpelt men der, fordi aktørere kun agere med spillet en gang i en tur og derefter bare får sit resultat af turen.

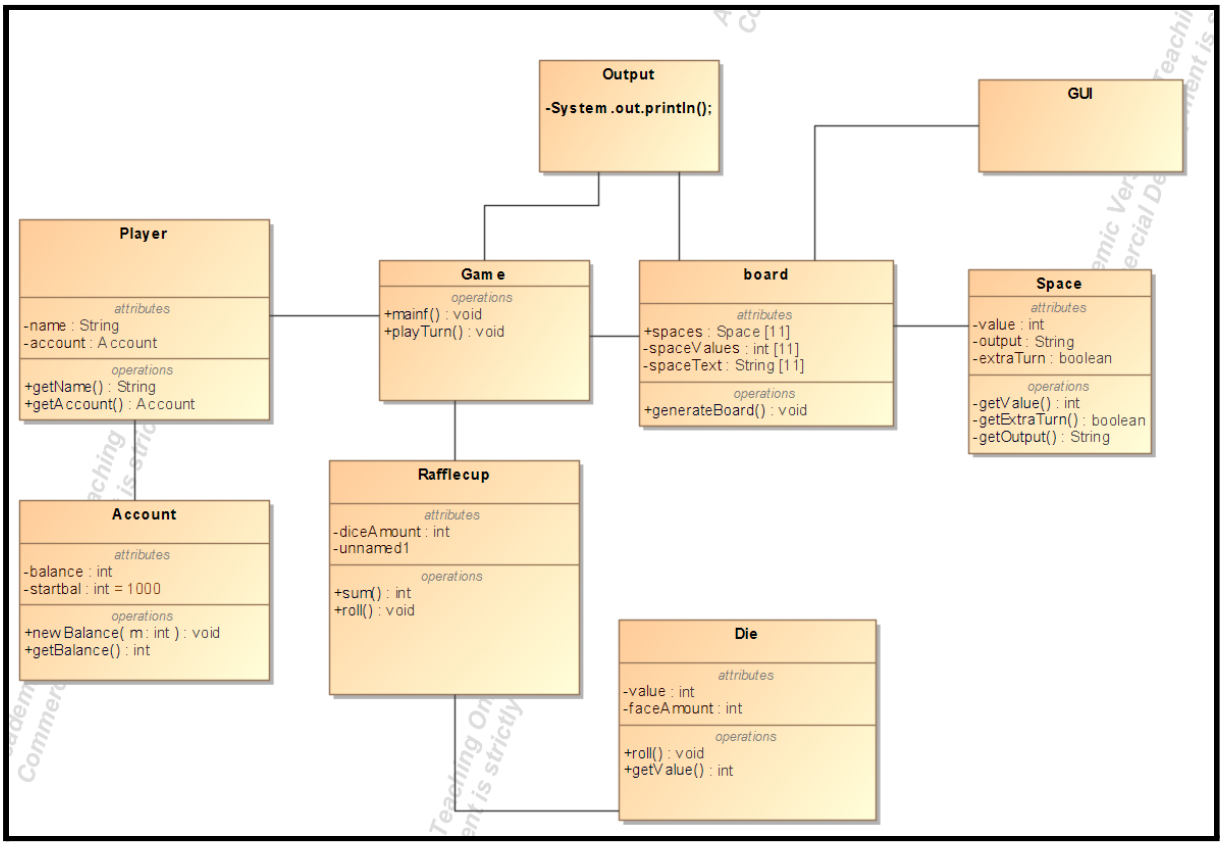


Domænemodel:



Design

Designklassediagram:

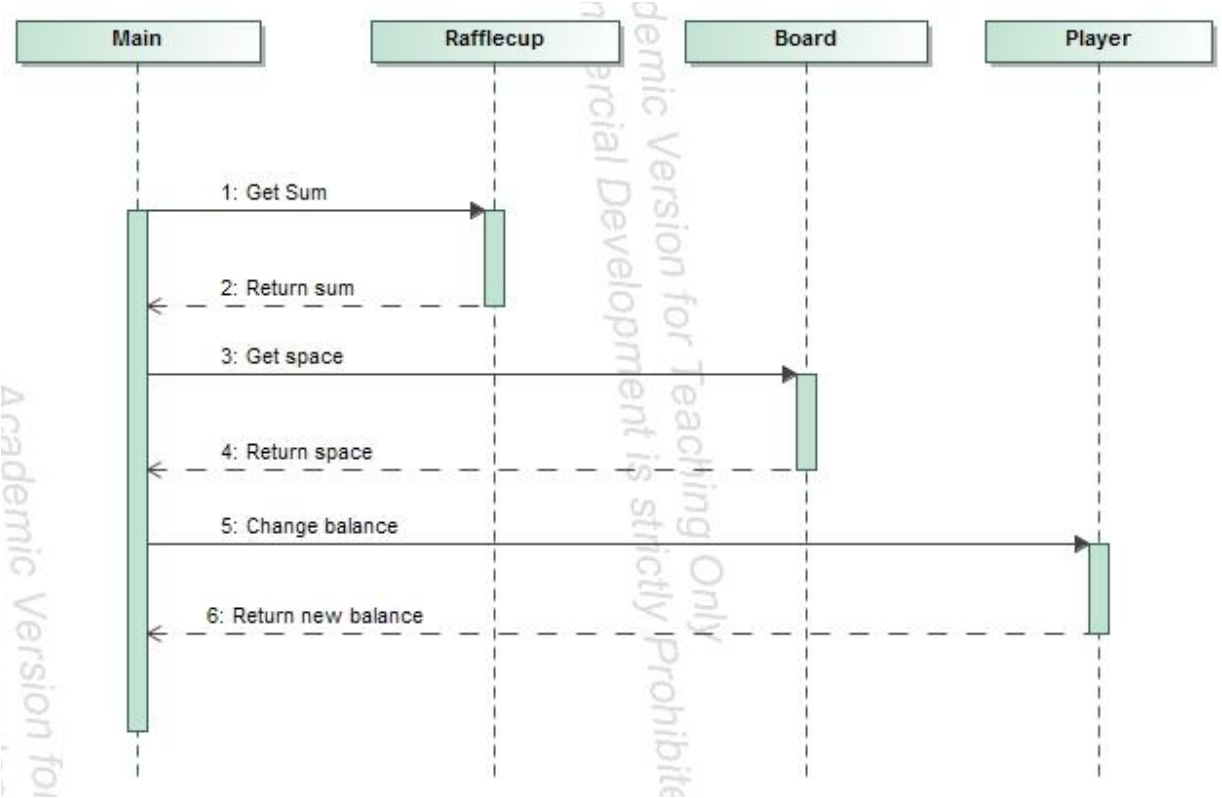


Ovenfor ses vores design klassediagram, i output klassen er mange metoder og også variable, dette har vi dog valgt ikke at inkludere da alle metoderne, og variablene bare udskriver, returner, eller indeholder en string. Desuden ses det også at vi ikke har fået implementeret GUI'en derfor er det svært at vurdere hvad dens tilhørsforhold er.

Sekvensdiagram:

Nedenfor ses et sekvensdiagram der beskriver samarbejdet mellem fire klasser i hver tur. Diagrammet bruger ikke klassen Output da den bare er til hver gang info kommer ud til brugeren. Diagrammet ses herunder. Det kan ses i diagrammet at Main styrer hvad der sker og kalder alle de metoder der ligger i en spil rundes forløb. Det ses at mange mindre metoder ikke er nævnt da de er lavet for at få disse hovedmetoder til at virke. Dette diagram giver en

forståelse af hvordan en tur i spillet fungerer. Så andre og en selv om nogle måneder har letter ved at forstå hvordan koden fungerer.



Anvendelse af GRASp mønstre :

Et mønster vi har lagt meget vægt på i vores kode er low coupling, dette kan også ses i vores design klassediagram. I vores kode betyder det at der ikke er coupling mellem klasser mere end højst nødvendigt. Alle klasserne er altså forholdsvis adskilt. Vi har også prøver at få high cohesion dette har vi gjort ved at lave klasser der kun gør enkelt ting. Dette kan for eksempel ses i Space, der er et objekt space og intet mere. Dette mønster ses i mange af vores klasser. Det ses også at vi har en creator klasse, altså board da den initialisere de objekter af space vi skal bruge. Det ses og at klassen Raffle Cup både er både er pure fabrication og indirection. Der ses indirection, da det skaber en nemmere interaktion mellem Main og Die. Dette ses ved at man skulle lave to terninger i main og hele tiden slå med begge to stedet for slår rafflebægeret med begge to for en. Det er pure fabrication, fordi spillet ikke behøver et rafflebæger for at man kan spille det.

Implementering

Under implementeringen vil vi ikke kommentere standard getter og sætter metoder. Vi har delt denne beskrivelse op i 5 segmenter. da vi har da vi har nogle klasse der hører tæt sammen disse 5 er

- Die og RaffleCup
- Account og Player
- Space og Board
- Output
- Game

Die og Rafflecup

Til ethvert terningespil skal der bruges en mængde terninger. Derfor har vi lavet en Die klasse med en to variable faceAmounts og value. Disse to variable beskriver henholdsvis antal sider på terningen og terningens nuværende værdi. I denne classe findes en konstruktor, der er laver en terning med et specifikt faceAmount. denne metode har en metode nemlig roll(); som giver en tilfældigt værdig. mellem 1 og faceAmount. Da der skal bruges mere end en terning har vi også lavet en rafflebæger klasse, der virker som en form for kontrollerklasse. Da den opretter og bruger Die objekter. Denne klasse har metoden sum der returnerer en sum har en given mængde terninger. Desuden kan man nemt skifte mængden af terninger, og antal sider pr. terning med metoden configure(); Så findes også Metoden roll(); og sum(); sum returnerer en int altså summen af slagene. og roll ruller terningerne.

Account og Player

Til dette spil skal vi også lave en account klasse, der holder styr på hver spillers pengebeholdning. Denne klasse indeholder balance og en startbalance. når man laver en account, skal man give en startværdi på balance. Her har vi brugt en konstruktor uden parametre så den altid giver en specifik startværdi da alle spillere starter med samme mængde penge i dette tilfælde 1000. Account har metoden newBalance(); denne metode tager en int som parameter og ændrer account.balance med denne int. Player klassen bruger dette objekt

som variabel samt navn. I denne klasse bruges en konstruktor der tager imod parameter navn som laver en ny account til en spiller.

Space og Board

Til vores spil har vi en board og space klasse, som holder styr på de felter/tal man kan lande på.

Space klassen består af de tre variabler som vi skal holde styr på, og som skal bruges når man slår det respektive tal. En værdi som er det antal point som spilleren enter mister eller vinder. En output tekst, som er den tekst der bliver skrevet ud. Og en boolean der bruges til at tjekke om man får en ekstra tur eller ej.

Board klassen bliver så brugt til at indeholde alle de her Space objekter som vi har, og er også der hvor vi opretter vores space objekter og assigner dem deres værdier.

Output:

Vi har lavet en klasse med output, denne klasses eneste funktion er at indeholde alle udskrifter koden bruger i en masse statisk metoder. Desuden indeholder den også nogle Strings der så er defineret derinde. Disse metoder er String metoder der returnerer en String eller en void metode der direkte printer noget.

Main

Til at styre hele vores spil og vores gamestates, har vi vores Main klasse, som er den der kontrollerer hele flowet af programmet.

Det er derinde fra at vi kontrollerer inputs fra spilleren, tjekker hvis tur det er og lader dem spille deres tur via en funktion og en boolean, ændrer balancerne på spillernes account, og holder styr på hvornår spillet slutter og hvem der har vundet.

Overordnet så kører vores spil via et while loop, som konstant tjekker efter hvert loop om vores win conditions er opfyldt, og spillet derved er ovre.

Hvis spillet ikke er ovre, så bliver der skiftet tur, og den næste spiller får lov til at rulle med terningerne og se hvad de lander på.

Test

Vi har lavet flere test til vores spil der tester sandsynligheder. De to mest central tests i dette spil er; om account klassen nogensinde statistisk vil ramme 0. Dette fungerer ved et loop der kører 100.000 gange. Her starter testen med en maks værdi på 1000 og hver gang værdien også ved at slå og derved få point øges maks. Hvis den nuværende pengebeholdning nogensinde bliver 1000 mindre end max fejler testen. Vi har kørt denne test flere gange og eftersom et spil er betydelig færre kast, burde det kunne vise nu. For at gå endnu videre vil vi beregne hvor langt tid et gennemsnit spil tager. dette ses i figuren nedenunder.

Felt	Penge	Gange man lander der gns	Penge pr 36 slag	Kast gns	Start penge
1	250	1	250	6,944444444	1000
2	-100	2	-200	-5,555555556	Penge for at vinde
3	100	3	300	8,333333333	3000
4	-20	4	-80	-2,222222222	Penge man skal anskaffe
5	180	5	900	25	2000
6	0	6	0	0	
7	-70	5	-350	-9,722222222	
8	60	4	240	6,666666667	
9	-80	3	-240	-6,666666667	
10	-50	2	-100	-2,777777778	
11	650	1	650	18,05555556	Gns. spillængde i slag
12		36	1370	38,05555556	52,55474453
	Her ses det statistiske gennemsnit for 36 terningkast				Grp. 2

Denne figur viser hvad vi forventer sker statistisk set. Et spil vil her tage i gennemsnit 52.6 kast pr. person. Når vi derfor prøver denne test med 100.000 kast kan men en ret stor sikkerhed vurdere at kontoen aldrig bliver under nul, hvis testen aldrig kommer under værdien maks-1000.

En anden Junit test, vi har lavet i vores program er en test der tjekker afvigelsen fra normalværdien på terningen. Denne test fejler hvis der er en afvigelse på over 1%. Vi har kørt denne test et par gange og har vurderet at terningerne er passende tilfældige til at det ikke påvirker spillet negativt. En anden test tjekker om vores configure metode virker som den skal. Den tjekker her om arrayets længde=det dice Amount man putter ind i metoden. Og om en tilfældig terning har det rette antal sider.

Konfiguration

Til at spille dette spil anbefales at der downloades den nyeste version af java. pt. jdk 17. Det kan sikker lade sige gøre med en ældre version, men herfra anbefaler vi den nyeste version af java.

Konklusion

Vi har nu lavet det ønskede program og lavet de ønskede artefakter. I analysen fandt vi frem til, hvad det var der blev efterspurgt, dette har vi gjort ved, at skrive en kravliste og et use case diagram over hoved use casen. Derefter har vi lavet en domænemodel der beskriver de domæner vores program indeholder. Vi har derefter designet, programmet ved hjælp af et uddybende design klassediagram. Vi har kodet spillet og testet use casen, så vi har en god ide om at spillet virker som det skal. Vi har også lavet de ønskede tekst, og talt lidt om det forventede statistiske udfald, og vist ved en stor test at det statistisk usandsynligt at en persons konto vil ramme 0. Vi har også lavet en test af de terninger vi bruger, og har vurderet dem til at være tilfælde nok til de ikke påvirker spillet negativt. Vi har til slut lavet et sekvensdiagram der viser hvordan en tur foregår, dette gør det nemmer at komme tilbage senere og se hvad ens program gør og hvordan de forskellige klasser arbejder sammen.

Litteraturliste

Larman Craig, Applying UML and Patterns, John Wait

Bilag:

https://docs.google.com/spreadsheets/d/13OCjaUVNp79IA-GldG9rL4mO49pN3H65FeYL56Z2OYQ/edit?usp=sharing