CDIO 2

Feltspil

Gruppe 36:
Jacob Johan Jørgens s175132
Mads Anton Schultz s175123
Monica Schmidt s175130
Nicolai Øyen Dam s175131
Nils David Rasamoel s165526

CDIO 2								
Time-regnskab	Ver. 2017-11-09							
Dato	Deltager	Analyse	Design	Impl.	Test	Dok.	Andet	Ialt
09-11-2017	Monica	3	2	0	0	3	0	8
09-11-2017	Nils	2	3	0	0	3	0	8
09-11-2017	Nicolai	2	2	0	2	2	0	8
09-11-2017	Mads	2	3	0	0	3	0	8
09-11-2017	Jacob	2	2	0	2	2	0	8

Indholdsfortegnelse

1. Indledning

Dette projekt har til formål, at programmere et spil mellem to personer, der på skift slår med to terninger. Spilleren rykker derefter frem til det felt, som terningernes sammenlagte værdi viser, hvilket kan være mellem værdier 2-12. Hver spiller starter med en pengebeholdning på 1000.

Spillet starter med, at en spiller slår med terningerne, spilleren rykker derefter frem til det felt, som svarer til terningernes værdi. Afhængigt af hvilket felt spilleren rammer, vil der komme en tekst, der enten har en positiv eller negativ effekt på spillerens pengebeholdning. Derefter vil det være den næste spillers tur. Spillet vil forsætte sådan, indtil en spiller opnår 3000 eller derover i deres pengebeholdning, den pågældende spiller har dermed vundet. Dog vil en spiller tabe spillet, hvis spillerens pengebeholdning bliver nul eller derunder.

Til analyse af projektet, er der anvendt kravspecifikation. Kravspecifikationen har formål at udspecificere de krav som programmet skal opfylde. Til dette er modellen FURPS+ anvendt, da FURPS+ kan bruges til at specificere funktionelle samt ikke-funktionelle krav, for et program.

Senere i analysen vil der blive lavet en use case, med et tilhørende use case diagram. Use Casen vil blive sammenholdt med de krav, fra FURPS+, som programmet skal opfylde.

Efterfølgende vil en domænemodel blive udarbejdet. Domænemodellen viser sammenhængen mellem de forskellige klasser i systemet, hvilket er vigtigt. Man kan derved sikre sig, at man har de klasser man skal bruge, og at sammenhængen mellem klasserne giver mening.

For at belyse interaktionen mellem de eksterne aktører og systemet er et systemsekvensdiagram blevet opstillet.

I forbindelse med designfasen i projektet, er sekvensdiagrammer anvendt. Sekvensdiagrammer er med til at vise, hvordan klasserne kommunikerer sammen i systemet hvilket udledes til specifikke handlinger. Yderligere vil der i designfasen blive anvendt GRASP-mønstre, der vil belyse klassernes ansvarsfordeling, Low Coupling og High Cohesion.

Ud fra den opstillede domænemodellen vil en design model blive udarbejdet. Med udgangspunkt i denne design model er koden til spillet Terningspil 2.0, blevet programmeret. Efter programmeringen af koden, er der blevet foretaget forskellige test af programmet for, at sikre at spillet opfylder de opstillede krav.

2. Analyse

2.1 Krav

Funktionelle krav

1. Begge terninger skal kunne kastes og skal have en tilfældig sum mellem 2-12. Dog skal de ikke vises til spilleren.
2. Programmet skal kunne oprette et spil til to spillere.
3. Programmet kan ikke oprette et spil med flere eller færre spillere end to.
4. Programmet skal kunne vise hver af de to spilleres point og opdatere dem, når der sker ændringer.
5. Programmet skal oplyse spillerne når den ene spiller har vundet. Dette sker når en af de to spillere som den første, opnår 3000 eller over i sin pengebeholdning.
6. Programmet skal oplyse spillerne, at en spiller har tabt spillet, hvis den ene spillers pengebeholdning går i 0.
7. Programmet skal oplyse spillerne om en enten positiv eller negativ virkning ved at lande på et unikt felt alt efter hvad summen af terningerne er. Eksempel: "Du er blevet voldtaget og forulempet, du mister 80 af din pengebeholdning". (Se Feltliste nedenunder)
8. De to spillere starter med 1000 hver i deres pengebeholdning.
9. Når en spiller slår summen 10 får spilleren en ekstra tur. Derudover gælder den pointmæssige konsekvens også som ved de andre felter. (Se Feltliste nedenunder)

Ikke funktionelle krav

Useability

n/a

Reliability

n/a

Performance

10. Programmet skal have lav responstid (< 0,333s)

Supportability

11. Koden skal være på dansk
12. Programmet skal være let oversætteligt til andre sprog.
13. Terningerne skal nemt kunne ændres, til en anden terning. (fx 8 sider osv.)
14. Programmets skal være indrettet sådan at "spilleren" og en given spillers "pengebeholdning" skal kunne bruges i andre spil.

Implementation

15. Spillet skal være på dansk
16. Spillet skal kunne spilles på DTU's databarer

Interface

17. Systemet skal fungere på computere, hvor den nyeste version af Java er installeret

Operations

n/a

Packaging

18. Programmet vil ikke have en fysisk indpakning

2.2 Feltliste

Som et krav til spillet, indeholder spillet en række felter, som spiller kan lande på når spillet spilles.

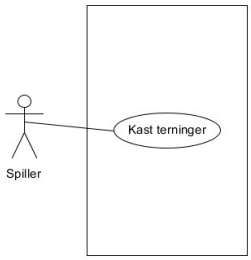
Sum	Navn på felt	Pris	Tekst der udskrives
2	Tower	+250	Du har fundet tårnet i Slagelse, og sælger 1g kokain for 250. Du er lidt rigere nu!
3	Crater	-100	Du har ødelagt dine sneaks, og blive nød til at købe nye for 100.
4	Palace gates	+100	Du er kunde nr. 100. Derfor får du 100.
5	Cold desert	-20	Du har brugt 20 på en is.
6	Walled city	+180	Du har vundet 180 I lotto. Tillykke med det, du gamle.
7	Monastery	0	Gud tilgiver dig, du behøver ikke betale.
8	Black cave	-70	Du har tabt mobile. Brug 70 på at fikse skærmen.
9	Huts in the mountain	+60	Sov I min hytte og du vil blive betalt 60 for din service
10	The Werewall (werewolf-wall)	-80	Du er blevet voldtaget og forulempet, du mister 80.
11	The pit	-50	Du fik en plads i pitten. Det koster 50.
12	Goldmine	+650	Du har fundet guld i bjergene og sælger det for 650, du er rig!

2.4 Use case

I dette projekt er der én use case, Kast terninger. Denne use case vil i de efterfølgende afsnit blive analyseret

Use case diagram

Her er der udarbejdet et use case diagram, hvor vi har tegnet aktøren (som er spilleren) og den ene use case der bliver udført, som er at kaste terningerne.



Aktører

Primær aktør:

- Spiller

Hertil menes der både spiller 1 og spiller 2.

Beskrivelse af aktør:

Spiller er en person, der har til formål at spille spillet.

Use case brief-beskrivelse

Her er der kort beskrevet hvad vores use case indebærer.

Use case	*Brief-beskrivelse*
Kast terninger	Spiller trykker på en knap og kaster terningerne, den sammenlagte værdi af øjnene beregnes.

Fully dressed use case beskrivelse

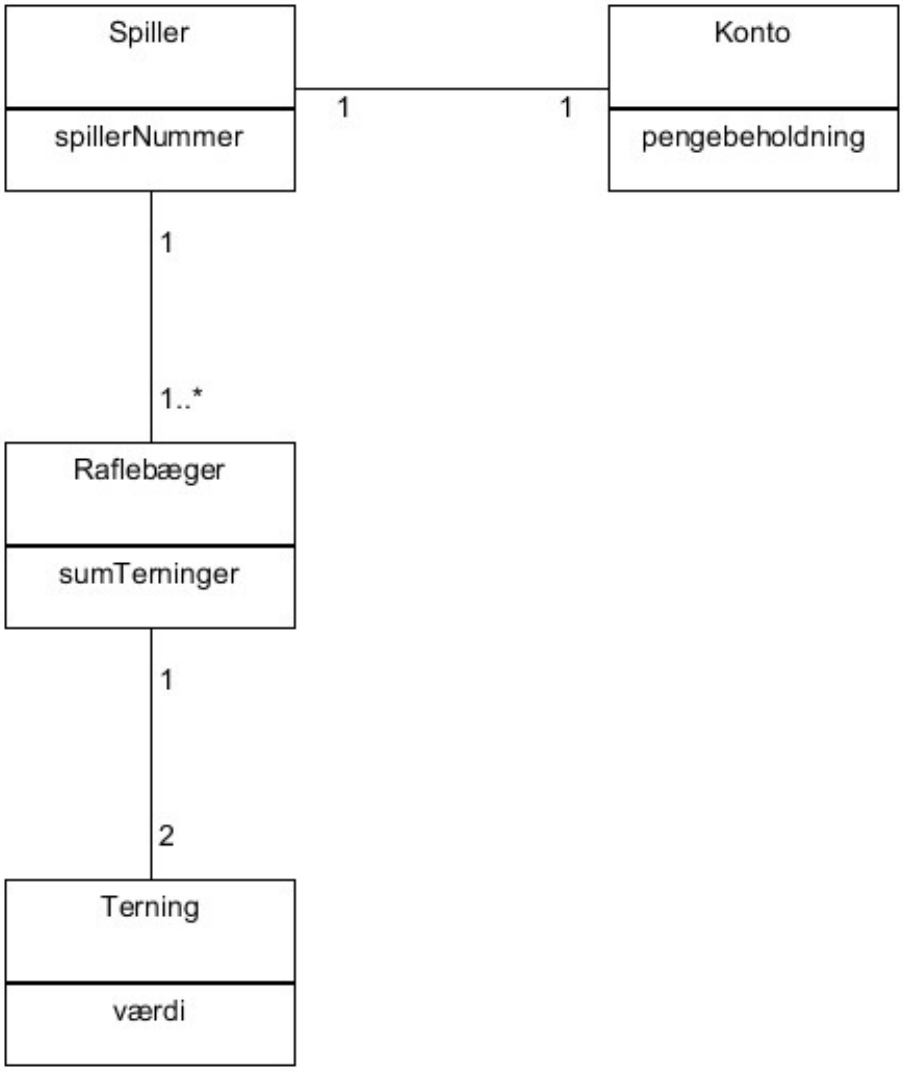
Vi har her udført en fully dressed use case beskrivelse, hvor vi går I dybden med den ene use case vi har. Den bliver beskrevet fuld ud, så den bliver helt gennemarbejdet.

Use case sektion	*Kommentar*
Use case navn	Kast terninger
Scope	Terningspil 2.0
Level	Det lykkes spilleren at kaste med to terninger og få en sammenlagt værdi.
Primær aktør	Spiller
Interessenter	DTU. Feltspillet skal kunne spilles på DTU's databare.
Forudsætninger	Det forudsættes at programmet skifter spiller, når en spiller har kastet med terningerne og spillerens pengebeholdning er blevet opdateret.
Succeskriterier	Terningerne kastes og den sammenlagte værdi af øjnene vises med det samme.
Hoved succes scenarie	1. Spiller kaster med terningerne. 2. Terningernes sammenlagte værdi vises. 3. Spillerens pengebeholdning opdateres, afhængigt af hvilket felt spilleren lander på. 4. Hvis en spillers pengebeholdning kommer over 3000, har spilleren vundet. 5. Hvis en spillers pengebeholdning kommer under 0, har spilleren tabt.

Udvidelser	1. (1.A) Spilleren kan ikke kaste med terningerne 2. (2.A) Terningernes værdi vises ikke 3. (2.B) Der er fejl i terningernes sammenlagte værdi 4. (3.A) Spillerens pengebeholdning opdateres ikke korrekt, ud fra feltets pågældende værdi. 5. (4.A) Hvis en spillers pengebeholdning kommer over 3000 og spilleren ikke erklæres som vinder. 6. (5.A) Hvis en spillers pengebeholdning kommer under 0 og spilleren ikke erklæres som taber.
Specielle krav	1. Når en spiller kaster med terningerne, skal den sammenlagte værdi vises inden for 0,3 sekunder. 2. Begge terninger skal have hele værdier mellem 1 til og med 6. 3. En spillers pengebeholdning opdateres afhængigt af det pågældende felt, som spilleren lander på.
Teknologi- og dataliste	1. (2.A) Der anvendes Math.random() metoden fra Math klassen i java.lang pakken. 2. (2.A) Der anvendes et GUI bibliotek, vises formål er, at vise terningernes værdi.
Optræden	Denne Use Case er den eneste i programmet, da spillet er afhængigt af, at to spiller kaster med to terninger. Værdien af terningerne udleder derefter de andre handlinger i programmet.

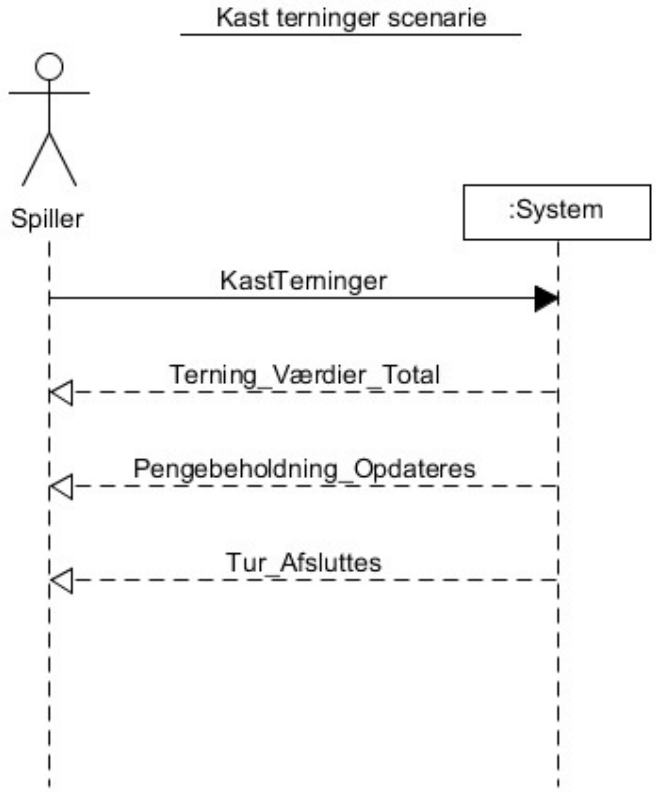
2.5 Domænemodel

Her har vi udarbejdet vores domænemodel, som beskriver hvordan vi tænker vores spil skal opbygges. Vi har altså 1 spiller, med en tilhørende konto. Dertil har vi så et raflebæger, som skal bruges til at slå med terningerne. Til sidst er der også en klasse der hedder terninger, som er der hvor vi generer de terningeværdier vi skal bruge.



2.6 Systemsekvensdiagram

Nedenfor illustreres use casen *Kast Terninger*, i et systemsekvensdiagram.



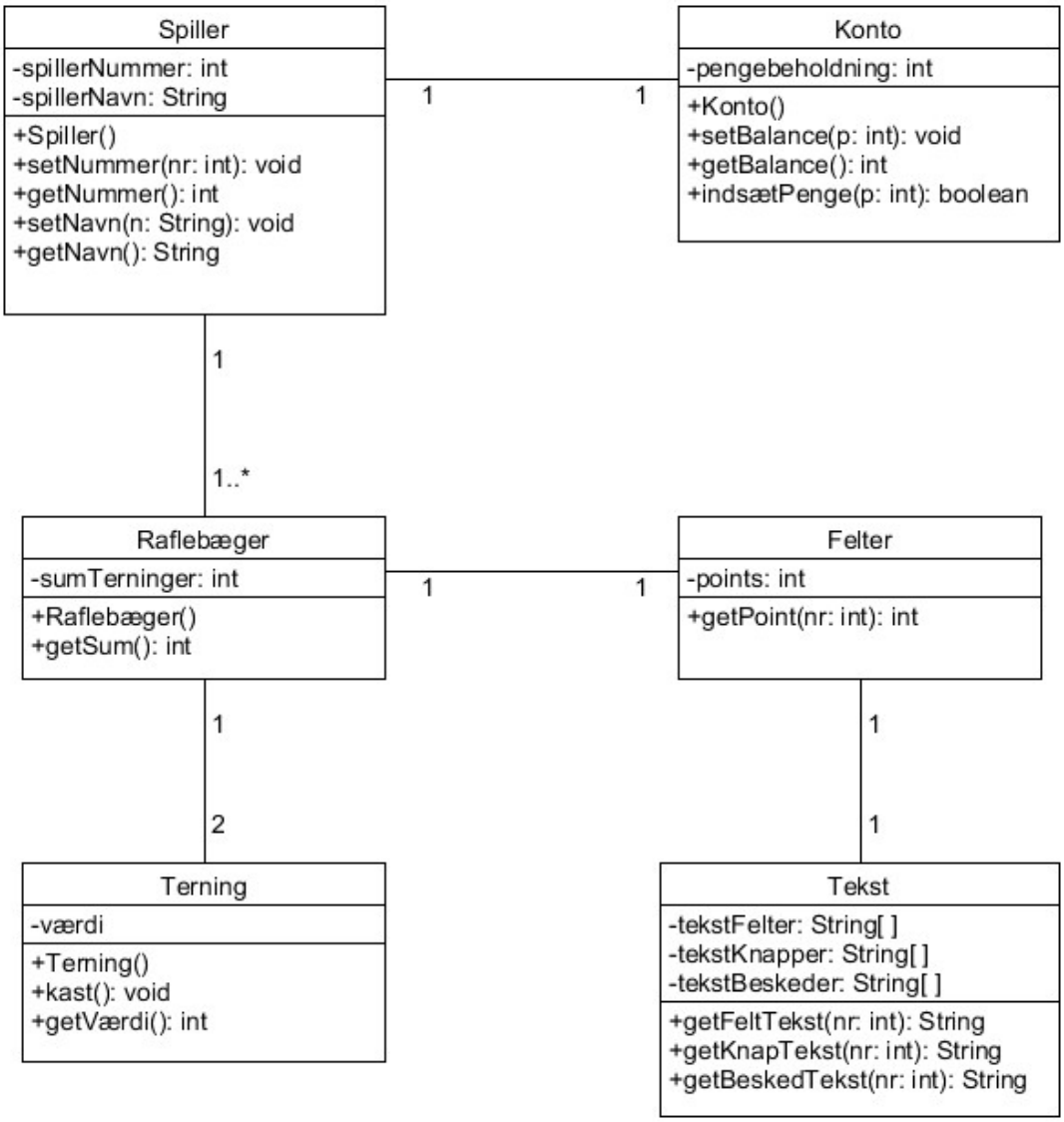
Kast terninger scenarie

1. Spiller starter spillet og opretter sig som spiller.
2. Spiller kaster med to terninger.
3. Den sammenlagte værdi af de to terninger vises.
4. Spillerens pengebeholdning opdateres negativt eller positivt, afhængigt af det pågældende felt.
5. Spillerens tur afsluttes.

3. Design

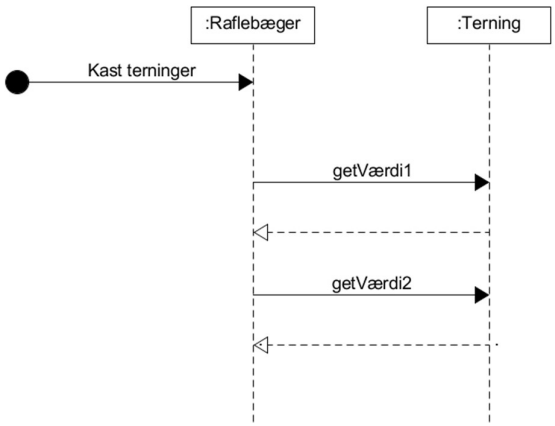
3.1 Designklassediagram

Design klassediagrammet viser resultatet af ansvarsfordeling. Her har vi også overvejet hvilke attributter og metoder vi skal bruge i de enkelte klasser. Så ud fra dette, har vi programmeret alle vores klasser.

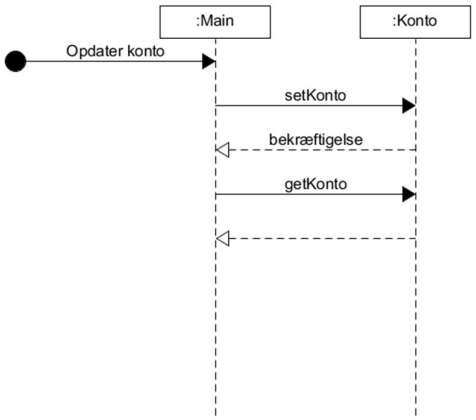


3.2 Sekvensdiagram

Sekvensdiagrammet viser kommunikationen mellem de to klasser Raflebæger og Terning, når en spiller foretager et kast. Kastet igangsætter klassen Raflebæger, der kalder værdi af terning 1 i terning klassen. Klassen Terning returnerer terning 1's værdi til Raflebæger klassen. Det samme sker mht. terning 2.



Dette sekvensdiagram viser hvad der sker, når klassen Main opdaterer kontoen for den spiller, der lige har slået. Main klassen ved, hvad der er blevet slået, og har fundet ud af, hvad der skal ske med spillerens konto. Spillerens konto sættes dermed til den nye værdi, og får en kvittering på den ikke er negativ. Derefter får den Main klassen værdien af kontoen, så kontoen kan opdateres.



3.3 Anvendte GRASP-mønstre

GRASP er en model der kan bruges til at fordele ansvar mellem objekter. Til at opdele ansvaret mellem objekter, kan man opdele deres klasser i tre kategorier:

Creator: Har ansvaret for, at oprette et objekt til en anden klasse.

Information Expert: Har ansvaret for at give de nødvendige informationer til at løse opgaven.

Controller: Har til ansvar at modtage og tildeler ansvar til et eller flere objekter, ved at koordinere inddata.

Information Expert

- Felter
  - Klassen Felter holder information omkring hvad felterne i spillet indeholder.
- Tekst
  - Klassen Tekst, indeholder al tekst til spillet.
- Konto
  - Klassen Konto administrerer spillerens balance under spillet.
- Terninger
  - Giver værdien til terningerne i spillet
- Spiller
  - Indeholder spillerens navn og nummer

Controller:

- Raflebæger
  - Klassen Raflebæger styrer klassen Terninger.
- Main
  - Klassen Main henter de forskellige informationer fra de respektive klasser for at kunne styre spillet.

3.4 High Cohesion og Low Coupling

I et system er det vigtigt at undersøge Low Coupling og High Cohesion, da dette kan være med at til forbedre ens kode.

Low Coupling kan være med til at sikre, at man har en lav afhængighed mellem klasserne, dette vil minimere påvirkningen af ændringer, samt er det med til at gøre koden lettere at genbruge og lettere at forstå.

High Cohesion kan anvendes til at sikre lav kobling mellem klasserne, High Cohesion sætter fokus på hvordan man bevarer objekter fokuseret og forståelig.

Ud fra det opstillede design klassediagram, kan det udledes, at klasserne i spillet har en Low Coupling og High Cohesion, da de fleste af klasserne har mellem to til en kobling med en anden klasse. Den klasse der har flest koblinger er raflebægeret. Dette ses ikke som et problem for programmet, grundet programmets størrelse og da det kun drejer sig om en klasse.

4. Implementering

4.1 Kode:

Her har vi lavet en opdelt forklaring til vores 3 pakker, som dette spil indeholder. Der vil være en forklaring til hver pakke, samt hvis der er nogle specielle ting ved de forskellige pakker eller klasser

Feltspil:

Dette er spillets første pakke, pakken indeholder alt det der får spillet til at køre. Denne pakke indeholder spillerklasse og en kontoklasse. Begge klasser er med til at holde styr på de ting, der har med spilleren at gøre, hvilket blandt andet styrer spillerens navn, spillernummer, pengebeholdning. I programmet indgår der yderligere to klasser, raflebæger og terning.

Raflebægerklassen har til ansvar, at hente to terningeværdier fra terningeklassen der lægges sammen, hvilket resulterer i spillerens terningeværdi. Terningeklassen har til opgave at slå med det antal terninger som raflebægeret siger den skal. På den måde bliver der generet tilfældige værdier mellem 2-12.

Feltklassen har til opgave at opdatere og styre spillerens pengebeholdning, når spilleren slår en bestemt værdi med terningerne. Feltklassen styrer dermed om kontoen skal ændres positivt eller negativt og med hvilket beløb. I Mainklassen styres hele spillet, her kontrolleres alt hvad der skal udføres i spillet.

Mainklassen er spillets controller og det er den klasse, der sørge for at hente de nødvendige informationer fra de andre klasser, så spillet kan udføres. Klassen Tekst har til formål at indlæse det valgte sprog, hvis filen findes i pakken, i nogle String arrays som man så kan hente teksten fra via metoden getTekst eller hente tilpasset tekst med input via metoden getTekstMedInput.

Klassen tekst har til formål at indlæse det valgte sprog, hvis filen findes i pakken, i nogle String arrays som man så kan hente teksten fra via metoden getTekst eller hente tilpasset tekst med input via metoden getTekstMedInput.

Klassen StrengArray har til formål at holde på en String Array og kunne håndtere fleksibilitet ved at kunne tilføje en String til for at udvide det via metoden tilføj. Den kan også returnere en String via indeks i array'et med metoden getFraIndeks.

Sprog

I denne pakke har vi lagt de to tekstfiler, der indeholder de 2 mulige sprog spillet kan bruges i. Dette er gjort for at gøre det lettere at oversætte spillet, til andre sprog. Vi har indtil videre kun lavet det i dansk og engelsk. Så det er her, vi importerer alle vores sætninger fra, som bruges i selve spillet.

Test:

Her ligger de klasser som vi har brugt til at teste de forskellige ting, der er beskrevet nede i test kapitlet. Der ligger to klasser, en til raflebægeret og en til mainklassen. Men alt dette er dybere beskrevet i test afsnittet.

5. Test

Vi vil nu udføre nogle forskellige tests, der skal sikre at vores spil virker som det skal, og er af god kvalitet. Derfor har vi opsat nogle forskellige test cases, som kan læses nedenunder.

Testcase

ID	Krav	Beskrivelse	Vejledning	Forventet output	Aktuelle output	Test date	Resultat
TC1	K2	Programmet skal kunne oprette et spil til to spillere.	Indtast navn -> Indtast navn	Der oprettes spil med to spillere	Der er blevet oprettet et spil med to spillere	7/11/17	Succes
TC2	K3	Programmet kan ikke oprette et spil med flere eller færre spillere end to.	Indtast navn -> Indtast navn	Der kan kun oprettes spil med to spillere	Det er kun muligt at oprette spil mellem to	7/11/17	Succes
TC3	K4	Programmet skal kunne vise hver af de to spilleres point, og opdatere dem, når der sker ændringer.	Indtast navn -> Indtast navn -> spiller1 kast -> spiller2 kast …	Vi forventer at vii får oplyst hvad der står på kontoen efter hvert kast	Vi får oplyst hvad der står på kontoen efter hvert kast	7/11/17	Succes
TC4	K5	Programmet skal oplyse spillerne når den ene spiller har vundet. Dette sker når en af de to spillere som den første, opnår 3000 eller over i sin pengebeholdning.	Indtast navn -> Indtast navn -> spiller1 kast -> spiller2 kast … -> Spiller X har vundet	Vi forventer vi får vi af vide når en spiller har vundet.	Vi får af vide når en spiller har vundet.	7/11/17	Succes
TC5	K7	Programmet skal oplyse spillerne om en enten positiv eller negativ virkning ved at lande på et unikt felt alt efter hvad summen af terningerne er. Eksempel: “Du er blevet voldtaget og forulempet, du mister 80 af din pengebeholdning”. (Se Feltliste nedenunder)	Indtast navn -> Indtast navn -> spiller1 kast -> få besked og antal point	Vi forventer, at får den rigtige besked og det forventede antal point.	Vi får den rigtige besked og det forventede antal point.	7/11/17	Succes
T6	K8	De to spillere starter med 1000 hver, i hver deres pengebeholdning.	Indtast navn -> Indtast navn -> spiller1 kast -> få besked og antal point	Vi forventer at en spiller starter ud med 1000 point.	En spiller starter ud med 1000 point.	7/11/17	Succes
TC7	K9	Når en spiller slår summen 10 får spilleren en ekstra tur. Derudover gælder den pointmæssige konsekvens også som ved de andre felter. (Se Feltliste nedenunder)	Indtast navn -> Indtast navn -> spiller1 kast -> få besked og antal point. indtil summen 10	Vi forventer, at ved en sum på 10 får spilleren en ekstra tur, og den pointmæssige konsekvens virker som normalt.	Ved en sum på 10 får spilleren en ekstra tur, og den pointmæssige konsekvens virker som normalt.	7/11/17	Succes

Junit-test

For krav K1 har vi valgt at lave en Junit test for at teste at terningerne der spilles med er tilfældige og at summen mellem dem kan give tal mellem 2-12. (Den ligger i testpakken) Ud fra testen ”testfordeling” kan vi se at terningen er tilfældig, fordi outputtet vil afspejle to normale terningers sandsynlige værdier.

På samme måde har vi med K6 lavet en Junit test, som tester at der bliver udskrevet false når beløbet bliver negativt. Hvis metoden indsætPenge tilbagegiver en false værdi, så vi der i vores Main, blive udskrevet at den pågældende spiller har tabt. Der

Test af terninger

Vi har for en sikkerhedsskyld oprettet en test klasse til raflebægeret. Vi vil gerne sikre os at alt hvad der sker med terningerne, er 100% tilfældigt. Så vi har opstillet nogle test, og som det kan ses, så er alle testene succesfulde. Så terningerne fungerer som de skal.

Test på databarer

For at være sikker har vi testet programmet på DTU’s databarer, hvor det er meningen, at spillet skal spilles. Det fungerer som forventet.

Test af negativ kontobeholdning

Vi har udført nogle Junit tests, som skal sikre at ligemeget hvad der sker, så kan en kontobeholdning ikke gå i minus. I den klasse som hedder ”MainTest” har vi lavet nogle tests, som tester om den kan blive negativ med 1 stor overførsel, og så tjekker vi også med forskellige andre negative overførsler. Men alle testene er succesfulde, da det på intet tidspunkt bliver muligt, at få en negativ kontobeholdning.

6. Brugervejledning

Konfiguration

Download mappen 36_del2, - Unzip mappen med WinRAR, 7-zip. (windows 10) - Højreklik på mappen. - Vælg ente WinRAR eller 7-zip -> Udpakfiler. - Extract mappen til ønsket destinaiton. Eksempelvis skrivebordet. C:\Users\bruger\Overførelser\ - Åben nu mappen. - Doubbelklik på filen feltspil.bat - Hvis fillen ikke kører: Højreklik -> kør som administrator. - Feltspillet vil nu åbne.

Brugervejledning

Start spillet ved at skrive navnet på spiller 1. Hvis spillet i stedet ønskes afsluttes, skal der blot skrives exit. Når spiller 1’s navn er skrevet, så skal der derefter skrives et navn på spiller 2. Når dette er gjort, starter spillet. Så vil spillet fortælle hvis tur det er til at kaste. Så skal der blot trykkes enter, for at kaste terningerne. Så vil spillet komme med nogle informationer, om hvad spilleren har landet på, og hvordan hans kontobeholdning ser ud efterfølgende. Så vil spillet igen komme frem og fortælle hvis tur det er til at kaste med terningerne. Sådan vil det forsætte, indtil spillet kommer med en vinder (hvis en spiller når 3000 point) eller en taber (hvis en spiller får under 0 point)

7. Konklusion

Vi kan konkludere at det er lykkedes for os, at løse opgaven. Vi skulle lave et spil der kunne spilles af 2 spillere, med nogle forskellige krav til systemet. Efter indledende arbejde, programmering og udførelse af forskellige test. Kan vi så sige nu, at det er lykkedes os at levere det ønskede spil.