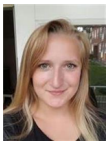
02312 02313 02315

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

CDIO del 2 - Goldmine

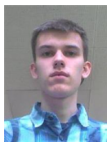
Hold A - Gruppe 17

s164177
Josephine Weirsøe



s185096
Jonatan Amtoft Dahl

s185118
Aleksander Lægsaard Jørgensen



s175561
Andreas Østergaard Schliemann

s185103
Søren Hother Rasmusen

9. november 2018

DTU Compute
Institut for Matematik og Computer Science

DTU Diplom
Center for Diplomingeniøruddannelse

Abstract

In this report we describe a dice game that we programmed. Our task was to create a dice game, which followed the specified requirements from the customer. The idea was that when you rolled the dice, you would lose or get some points and in the end there would be a winner. This report touches on both how the software we wrote works and shows various diagrams on different aspects of the project. Furthermore, this report also gives a description of all the different tests that we made to ensure that the program works in the way we expect it to. In the end we can conclude that our program works, and the work process was rather smooth.

Indhold

1 Indledning

Projektet består af et terningsspil mellem to spillere, som skiftevis slår med to terninger, herefter lander de på et felt der har en konsekvens der påvirker deres penge/point-beholdning. begge spiller starter med 1000 points og vinderen er den spiller der først når 3000. I projektet har vi udarbejdet nogle artifacts som beskriver programmets opbygning og cases, her fremgår blandt andet hvilke klasser og forskellige metoder der benyttes. Produktet skal bruges på computerene i DTU's databare. Programmet er skrevet med programmeringssproget Java i udviklingsprogrammet IntelliJ.

2 Timeregnskab

Da der ikke er blevet gjort klart på noget tidspunkt hvad et timeregnskab er eller bør indeholde, har vi lavet en meget dybdegående tabel. I nogle tilfælde er vi flere der har siddet med samme opgave og arbejdet på det sammen. Generelt har det været tilfældet at hele gruppen har siddet sammen og arbejdet det sammen antal timer, selvom der selvfølgelig er 6 nogle ting der er blevet arbejdet på hver for sig.

Opgave	**Josephine**	**Aleksander**	**Søren**	**Jonatan**	**Andreas**
Generelt LaTex	Mange timer				
Forside	120+ min				
Timeregnskab	75 min	20 min			
Abstract og indledning	20 min				
Vision			60 min		
Krav	60 min	120 min			20 min
Sekvensdiagram		330 min		95 min	17 min
Design klassediagram					63 min
Design klassediagram ny			120 min		
Domænemodel			50 min		
System Sekvensdiagram			90 min		
Aktører			20 min	35 min	
Use case	100 min			35 min	70 min
Projektplanlægning	40 min				
konklusion				35 min	
Implementering				50 min	
Positive og negative test				110 min	
Kode: UI				360 min	
Kode: Generel kodning		250 min			
Kode: User Branch	80 min		80 min		
Kode: Game/Goldmine klassen		355 min			
Kode: Die klasse					100 min
Kode: Test afsnit + Junit			240 min		40 min

3 Hovedafsnit

3.1 Krav

3.1.1 Vision

Kunden vil gerne have et terningespil, der spilles mellem to spillere. Spillet skal kunne spilles på DTU's databarer, uden bemærkelsesværdige forsinkelser.
Spillerene skiftes til at slå med 2 terninger og lander på forskellige felter. Hvert felt har forskellige navne og værdier, som har indflydelse på spillerens fremgang (se Feltliste). Hver spiller har en pengebeholdning, som starter på 1000, og den første spiller, som opnår 3000 vinder spillet.
Kunden vil desuden have at spillet skal kunne oversættes til andre sprog, og at man skal kunne skifte terninger. Kunden vil også gerne kunne genbruge dele af programmet til andre spil. Mere præcist spilleren og hans pengebeholdning.
Vi har valgt at henvise til spillet som "Goldmine", så det er det som det bliver kaldt i denne rapport.

Kunden ønsker desuden at kunne se hvert enkelt gruppemedlems bidrag ved hjælp af et timeregnskab og ved at aflevere et GIT-repository over commits.

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

3.1.2 Kravlister

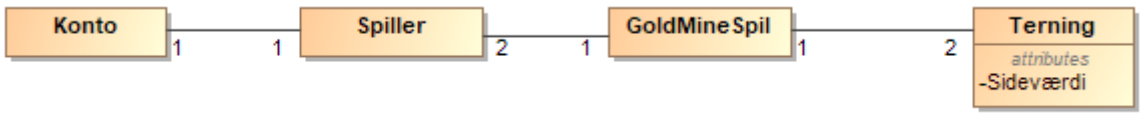
Nedenfor ses to lister over de krav der stilles til programmet delt op i funktionelle og ikke funktionelle krav, samt en vurdering af deres vigtighed.

Funktionelle krav	**Prioritering**
Systemet skal indeholde to spillere	Mest vigtig
Systemet skal kunne slå med to terninger hvor udfaldet er mellem 2 og 12	Mest vigtig
Systemet skal kunne lande på 11 forskellige felter med hver deres effekt	Mest vigtig
Systemet skal udskrive en tekst omhandlende det aktuelle felt	Mest vigtig
Systemet skal uddele spillerne med en pengebeholdning på 1000 til start	Mest vigtig
Spillet skal slutte når en spiller når 3000 points	Mest vigtig
Spillet skal let kunne oversættes til andre sprog	Nogenlunde vigtigt
Spillet skal let kunne skifte terningertyper	Nogenlunde vigtigt

Ikke funktionelle krav	**Prioritering**
Skal kunne køres på DTU's maskiner i data barene	Mest vigtig
Spilleren og hans pengebeholdning skal kunne bruges i andre spil	Mindst vigtig
Der skal være en GUI	Mindst vigtig
Systemet skal benytte terninger fra CDIO_1	Mindst vigtig

3.1.3 Domænemodel

I en domænemodel vises sammenhængen mellem de vigtigste begreber i et program. I denne domænemodel vises spillet Goldmine. Spillet Goldmine spilles mellem 2 spillere, med 2 terninger og hver spiller har en score eller "konto". Derfor er vores begreber "Spiller", "Terning", "Konto"og selve spillet selv. I Figur 1 ses sammenhængen mellem disse begreber.



Figur 1: Domainemodel

3.2 Analyse

3.2.1 Aktører

Systemets aktører

Goldmine er et simpelt spil, som ingen andre systemer interagerer med. Derfor er der kun 2 aktører:

1. Bruger
2. Kunde

Aktørernes formål og type

Formålet for brugeren, er at kunne spille spillet, sammen med en anden person i databarene på DTU. For kunden er formålet at have et spil, som kan spilles i DTU's databarer. Desuden vil kunden gerne kunne genbruge dele af koden til senere udviklinger. Dette gør at kunden er sat, som en Offstage aktør.

Aktørtype	**Aktører**
Primære	Bruger
Supporterende	-
Offstage	Kunde

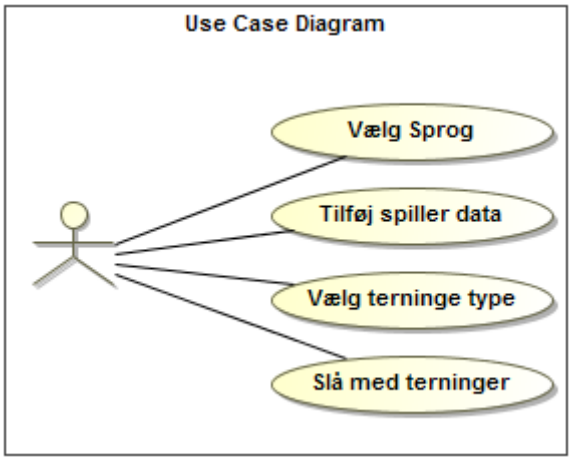
3.2.2 Use cases

liste over use cases

1. Slå med terninger
2. Indskriv spilleroplysninger
3. Bestem terningtyper
4. Skift sprog

Use case diagram

Selvom vi har endnu en aktør end hvad vi har indskrevet i vores use case diagram på figur 2, så er det ikke vist i use-case diagrammet. Vi har valgt at gøre dette, da vores "kunde"aktør ikke har en direkte forbindelse til nogle af vores use-cases. Dette er fordi kunden ikke rigtigt skal bruge selve spillets funktionalitet, men kun er interesseret i at spillet kan findes på DTU's databarer.



Figur 2: Use Case Diagram

Brief-udgave af use cases

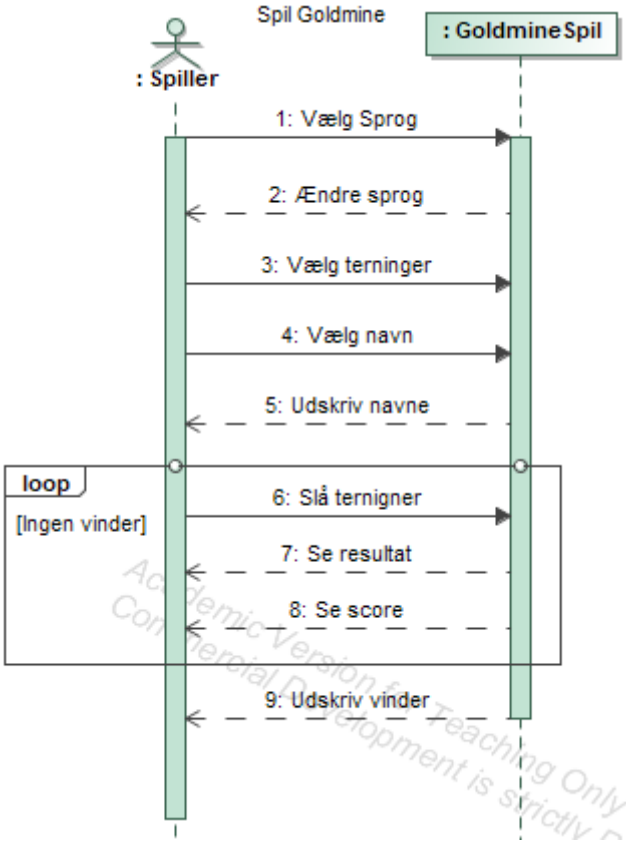
1. Slå med terninger: Spilleren slå med to terninger.
2. Indskriv spilleroplysninger: Spilleren indskriver deres navn.
3. Bestem terningtype: Spilleren skriver, hvor mange sider der er på teringen.
4. Skift sprog: Spiller skifter sproget, som programmet udskriver teksten i.

Fully dressed af use case: Slå med terninger

Use case: Slå med terninger
ID: 01
Brief description: Spilleren vælger at slå med terningerne, programmet udskriver antallet øjne der er på terningerne. Derefter udskrives der hvilket felt der er ramt, samt hvilken effekt dette har.
Primary actors: De to spillere
Secondary actors: Kunden
Preconditions: Der er valgt sprog, terninger og indtastet spiller data.
Main flow: 1 Spilleren slår med terningerne. 2. Programmet udskiver konsekvensen heraf. 3. Spiller to slår med terningerne. 4. Programmet udskiver konsekvensen heraf. 5. Gentag trin 1 til 4 indtil der er fundet en vinder. 6 . Programmet udskriver hvem der har vundet.
Postconditions: Programmet skal kunne gemme spillerenes data og deres points.
Alternative flows: Ingen.

3.2.3 System sekvensdiagram

Et system sekvensdiagram er lavet over hele spillet "Goldmine"(se Figur 3). Normalt ville man lave et system sekvensdiagram over en enkelt essentiel use case, men da alle vores use cases er meget simple, er der lavet et samlet system sekvensdiagram over hele programmet. Diagrammet viser hvordan vores aktør "spiller"kommunikerer med systemet og hvilke output "spilleren"får.



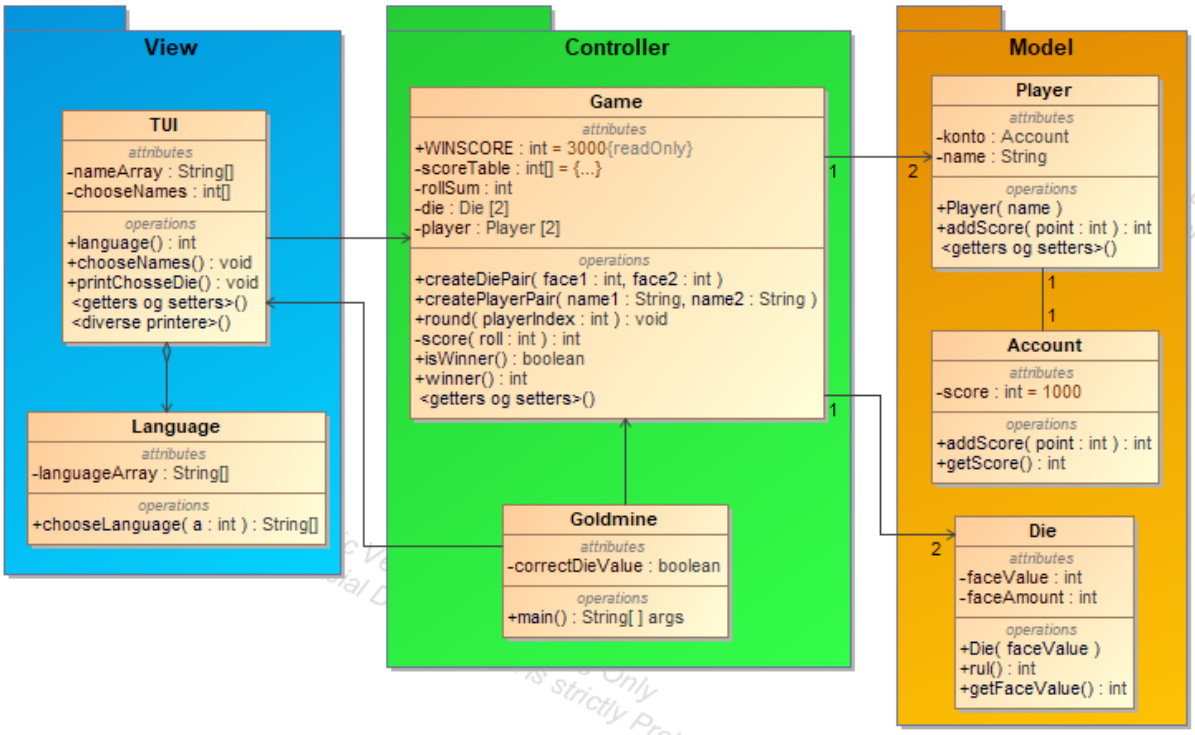
Figur 3: System sekvensdiagram

3.3 Design

3.3.1 Design klassediagram

Design klassediagrammet viser de reelle klasser, som vi har tænkt os at implementere, deres metoder og hvordan de relaterer til hinanden. Det originale design klasse diagram kan ses i bilag som Figur 11.
I løbet af implementeringen har vi oprettet andre klasser end originalt forventet, så vi har lavet et nyt design klassediagram (Figur 4, der viser de klasser, sammenhænge, metoder og attributter, som der reelt er i programmet. Dette design klassediagram bruges til at give et overblik over hvordan programmet er opbygget og virker. Endvidere er klasserne lagt i pakker, for at vi under implementeringen kunne holde et overblik over, hvilke klasser havde hvilke ansvarsområde.

I løbet af hele projektet er GRASP principperne blevet brugt. I sær principperne om høj samhørighed og lav kobling har der været stor fokus på. Dette er gjort for at lettere kunne genbruge dele af programmet, som fx Player og Account klasserne og for at lettere kunne vedligeholde programmet, hvis dele af det ikke virker.



Figur 4: Design Klassediagram

3.3.2 Sekvensdiagram

Det originale sekvensdiagramm ses på figur 6 og ligger under bilag.
Efter koden var skrevet, besluttede vi os for at reviderer sekvensdiagrammet, ved at dele den op i fire nye sekvensdiagrammer, som kan findes som bilag. Navnene på disse nye diagrammer samt deres figurnummer i bilaget er som følger: "vælg sprog"er figur 7, "vælg terninger"er figur 8, "vælg navne"er figur 9 og "spil spil"er figur 10.
Det originale sekvensdiagram blev lavet før koden var skrevet, og var derfor baseret på hvordan det oprindeligt var tænkt at koden skulle struktureres. Efter vi begyndte på koden, indså vi, at det ville være mere komplekst at skrive end som så, og de fire nye diagrammer blev lavet, for bedre at repræsenterer koden.
De fire nye sekvensdiagrammer er baseret på vores fire use-cases, hvilket også stemmer bedre overens med hvordan sekvensdiagrammer skal udtænkes. Det originale diagram var på en måde et sekvensdiagram over alle vores use-cases på samme tid, hvilket IKKE er formålet med et sekvensdiagram.
I de fire sekvensdiagram er TUI-klassen vist, men det er ikke alle de metoder som kaldes i koden som er vist i diagrammet. TUI-klassen har mange metoder som viser brugeren hvad der foregår. Nogle af disse klasser kalder også Game-klassens get-metoder for at beskrive præcist hvad der sker og på hvis tur osv. Sekvensdiagrammerne ville hurtigt blive uoverskuelige hvis alle metoderne som giver spillerne feedback skulle indskrives, så de metoder som kun bruges til at udskrive feedback til brugeren er udeladt.

3.4 Implementering

Under implementering prøvede vi så vidt som muligt, at følge vores diagrammer, så vi fik de samme klasser i koden som i vores diagrammer. Dog viste det sig at under implementeringen af koden, at vi blev nød til at tilføje nogle flere klasser, og at vi blev nød til at lave nogle af klasserne lidt om. Derfor har vi ændret i nogle af vores diagrammer for eksempel design klassediagrammet, så det repræsenterer

vores system korrekt. Vi valgte at gå tilbage og rette i vores diagrammer, fordi kodning er en agil og en iterativ proces.
Vi brugte "big bang"til at sætte alle delene af programmet sammen. Under selve kodningen blev individuelle metoder testet, men programmet blev ikke testet som helhed før til allersidst. Da vi brugte metode i slutningen af kodningsprocessen, fungerede den rigtig godt. Dette var fordi selvom big bang metoden har ret stor risiko, for at gå galt, så er vores system forholdsvis småt.
Kigger vi videre i forhold til 3-ugers kursuset, vil den største ting vi kan tage med fra dette projekt i forhold til implementering være, nok ikke at bruge big bang metoden. Det vil give mere mening at test programmets individuelle dele med stubklasser, så det vil være nemmere at isolere fejl og rette dem hurtigt.

3.5 Test

Kunden har bedt om at programmet er testet og at disse test kan gentages. Kunden har bedt om en test, som undersøger, om spillerens pengebeholdning kan blive mindre en 0.
Kunden har også bedt om andre relevante test.

Ligesom i CDIO1 er terningernes funktionalitet fundamental for at spilleret kører korrekt. Derfor er flere af JUnit testene fra CIDO1 blevet genbrugt for at undersøge terningene. Se Unit test afsnittet.

3.5.1 Positiv test

Vi vil her tjekke om de funktionelle krav som programmet har er blevet opfyldt. Derfor kører vi igennem hvert krav, for at se om spillet opfylder den opgave, vi har fået stillet.
Vi kunne teste her i de positive test, om en spillers score kan komme under 0. Dog har vi lavet en unit test, som tester præcis dette, derfor vil vi ikke lave nogle positiv test, for at se om dette kan ske.

Antal spillere
Den første positive test tester om spillet har to spillere som kan spille spillet. Med denne test kan vi se, at det er det som er sket, og at spillet i starten spørger om præcis to navne, så derfor er der to spillere, se figur 12 i bilaget.

Start point
Denne test tester om en spiller starter spillet ud med 1000 point. I bilaget kan det ses, at på spillerens første tur, hvor han fik 0 point, har han stadigvæk 1000 point. Dermed må spillerne starte med 1000 point, se figur 13 i bilaget.

Slut point
I den her test undersøger vi om spillet afslutter og kårer en vinder. Dette kan ses i dette bilag, at det er tilfældet, når en spiller når 3000 point eller mere, se figur 14 i bilaget.

Forskellige effekter for forskellige slag
Her undersøger vi om hvert forskelligt slag man slår, har en forskellig effekt, så effekten af ens tur er afhængig af hvad spilleren slår med terningerne. Dette var en succes, fordi hvert tal man kan slå har et andet udfald, se figur 14 i bilaget.

Forskellige tekster for forskellige effekter
Denne test tjekker bare om der kommer tekst for det slag man slår og hvad man finder i spillet, se figur 14 i bilaget.

Ekstra tur Testen her undersøger, om man efter et slag hvor man slår 10 får en ekstra tur. Vi undersøgte dette og det viser sig at fungere, da Jonatan i dette tilfælde, slår to gange i træk og det derefter

er Sørens tur igen, se figur 15 i bilaget.

Forskellige slags terninger Vi tester her om man kan vælge forskellige terninger, og om de terninger man vælger er indefor det rigtige rammer, da man ellers ville kunne slå noget, som der ikke er givet effekt til. I dette bilag ses det at man skal vælge de terninger man spiller med, og at systemet spørger en igen, hvis de terninger man vælger, er udenfor det rigtige interval, se figur 16 i bilaget.

Skift sprog
Testen her undersøger om det er muligt at skifte sprog, som spillet skal spilles på. Og igen viser vores test at det kan man, se figur 17 i bilaget.

3.5.2 Negativ

I vores program er der ret få steder hvor systemet skal bruge et specielt type input fra brugeren, før at det kan regne på de ting. Disse tre er når man vælger sprog, terninger og navne. Derfor har vi prøvet at teste med storetal, småtal, tegn og bogstaver, dog giver spillet ingen fejl. Dette er fordi vi har indsat exceptions, som kan gøre at fejlene ikke kommer til at ske. Andre steder i programmet hvor der kommer input fra spillerne, er når systemet pauser og venter på, at en spiller klikker Enter til at rulle terninger eller andet. Her kan man også skrive hvad man har lyst til, uden at der kommer fejl.
I vores CDIO1 projekt havde vi et problem som vi ikke kun fikse, det var når vi skrev f.eks 'e' som input, så ville den ikke skrive hvad der var inde i gåsøjnene. Så hvis man skrev et input sådan her H'e'j så vil programmet kun registrer Hj. Dette er dog ikke tilfældet denne gang, vi ved ikke hvad der har fikset det problem. Fejl kunne både være i vores kode, eller en bug i IntelliJ eller måske i Java selv.

3.5.3 Unit test

Til testning af koden er der også blevet brugt JUnit. Dette er gjort, så vigtige dele af testen let kan gentages af kunden med (forhåbentlig) samme resultat.

Alle test i DieTest-klassen er lavet med terninger med 6 "sider". Testen "Kast"undersøger om en terning kun giver slag mellem 1 og 6, og at alle slag mellem 1 og 6 bliver slået.
Testen "En_Terning"undersøger om alle mulige slag er ligeligt repræsenteret med en fejlmargin på 4%.
Testen "To_Terninger"undersøger om summen af 2 terningeslag vil danne en normalfordeling, som ville være forventet med 2 terninger. Dette blev undesøgt med en fejlmargin på 5%.
Testen "Ens_Slag"undersøger om terningernes slag kan være ens med en fejlmargin på 4%.

Alle disse 4 test er lavet, for at undersøge om de terninger, der er programmeret opfører sig tilfældigt og naturligt. Eftersom at alle testene kom tilbage bestået, så kan vi konkludere at terningerne virker og opfører sig naturligt.

Der er også lavet en JUnit test, der undersøger om spillerens score kan bliver mindre end nul. Dette er gjort ved at lave en stubklasse, som kun kan gøre spillerens score mindre. Derefter er der oprettet en konto, som er blevet fratrukket mere end alle dens points grundet stubklassen og til sidst er det undersøgt om kontoens score er lig nul. Da denne test kom tilbage positiv, kan vi konkluderer at spillerens score ikke kan blive mindre end nul.
Der var også påbegyndt en test, som ville undersøge om en spiller ville vinde ved 3000 points. Dog blev det fungerende spil færdigt før testen var færdig, så testen blev irrelevant og er derfor ikke lavet færdig.

3.6 Projektplanlægning

Planlagt forløb
Vi har i projektet planlagt at starte med at opstille kravene, use cases og opstille samtlige digrammer og modeller. Efter at have lavet forarbejdet er det planlagt at begynde på koden, hvor der skal laves

klasser, metoder og konstruktører som til sidst skal sammensættes til spillet vi er blevet bedt om at lave. Hvis der er overskud til det vil vi indføre GUI'en men dette er dog ikke en prioritet. Koden skal løbende testes, det er blandt andet planlagt at benytte JUnit. Planen er desuden at koden skal være tilgængelig gennem github for kunden og gruppens medlemmer.

Reelt forløb
Efter at være startet på projektet har det fungeret godt med at starte ud med opstilling af krav og use cases. Det har desuden givet et godt overblik at starte med at lave diverse diagrammer, på trods af at der opstod nogen tvivl om hvad der var relevant. Det forbyggende arbejde har gjort det nemt at starte på kodningen. Da vi startede på koden gik det forholdsvist nemt, da man ud fra diagrammerne nemt kunne se hvilke klasser osv. der skulle være og hvad disse skulle indholde.
Da vi kom længere ind i kodningsprocessen fandt vi ud af, at ikke alle vores diagrammer var lige realistiske. Derfor var vi nød til at gå tilbage, og redigere lidt i vores diagrammer, for at få koden og diagrammerne til at stemme overens.
Vi fik ikke implementeret den GUI som vi fik udleveret, vi fik dog lavet en anden som var baseret på print statements.
Den resterende tid og afslutning af rapporten samt kodningen gik godt, med godt samarbejde. Så alt i alt forløb vores projekt godt, og vi fik lavet alt til tiden uden at have en masse stress til sidst.

4 Konklusion

Under dette projekt har vores proces været god, dog har forløbet ikke været helt som planlagt fra starten af projektet. Vores proces i starten byggede meget på at vi først skulle komme igennem alle diagrammer og så kode, skrive test og lignende til sidst. Vores proces endte med at være meget mere flydende, hvor vi kiggede tilbage på vores diagrammer, og lavet om på dem, hvis de viste sig ikke at fungere i praktisk. Selvom der har været forholdsvis meget at lave mod slutningen af forløbet, har vi kunne arbejdet koncentreret og fået lavet det, som skulle laves i nogenlunde tid.

Vores produkt kom til at være lidt over vores forventninger, vi fik nemlig implementeret exceptions, som har gjort at mange af de fejl vi havde i nogle forrige projekter ikke længere var til stede. Programmet udfyldte alle de krav som var sat fra starten af. Selve koden er bedre skrevet og mere overskueligt, end vi havde regnet med i starten.

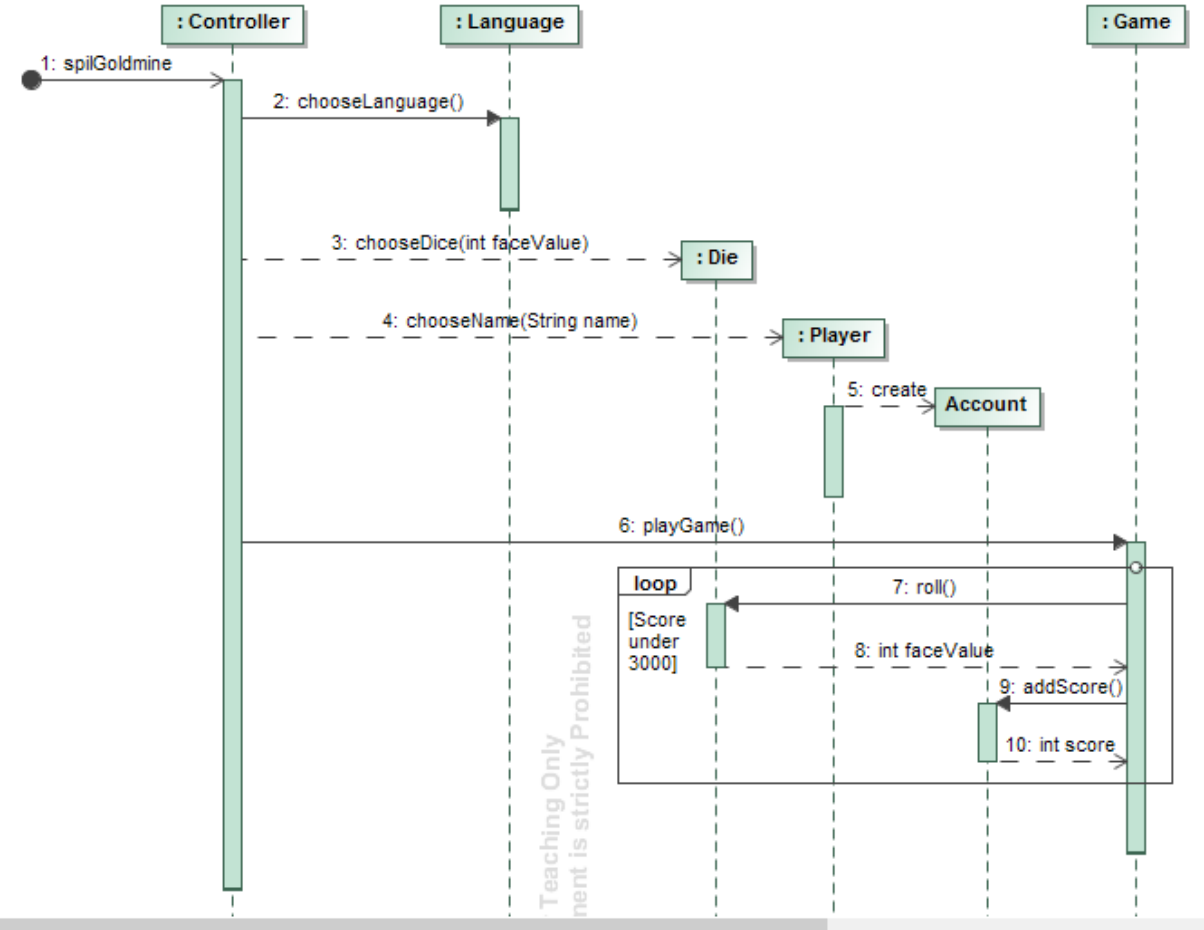
Vores CDIO2 projekt har været markant mere overskueligt både i rapporten, med diagrammer og lignende, men også med koden i forhold til vores CDIO1 projekt. Denne overskuelighed er kommet fra mere fokus på high cohesion og low coupling, som har gjort hele systemet mere lige til, mere åbentlyst og nemmere at finde rundt i. I dette projekt har vi fokuseret meget på at have et godt timeregnskab til sidst, det har vi ikke haft i tideligere projekter, så her har dette projekt også været bedre.
Vi har kunnet bruge git og github bedre, og har brugt vores branches mere effektivt, uden at skulle skifte branch hele tiden, hvilket var et større problem i CDIO1.

Programmet selv virker som det skal uden fejl og alle de vigtigste krav er blevet opfyldt.
Vi har lavet en requirement traceability matrix som viser hvilke af vores use-cases opfylder hvilke af vores krav. Som man kan se, er alle krav blevet opfyldt af en eller flere use-cases, og alle use-cases har opfyldt et eller flere krav.

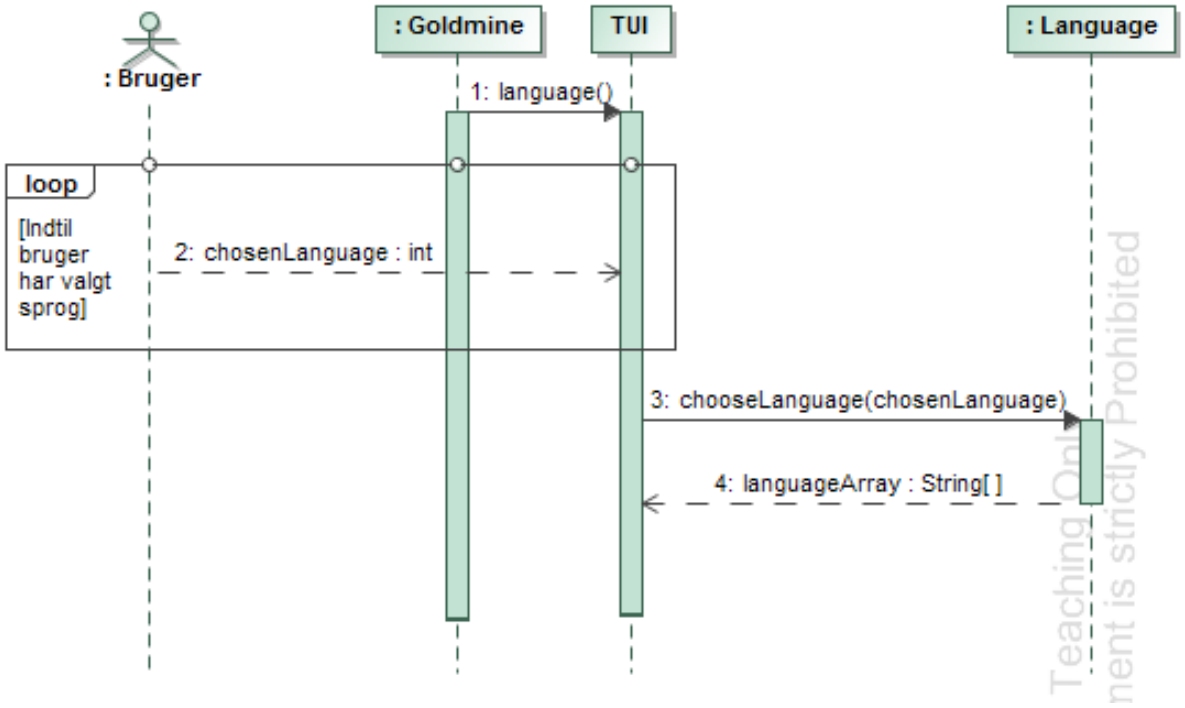
	Vælg Sprog	Tilføj spiller data	Vælgterninge type	Slå med terninger
Systemet skal indeholde to spillere				
Systemet skal kunne slå med to terninger hvor udfaldet er mellem 2 og 12				
Systemet skal kunne lande på 11 forskellige felter med hver deres effekt				
Systemet skal udskrive en tekst omhandlende det aktuelle felt				
Systemet skal uddele spillerne med en pengebeholdning på 1000 til start				
Spillet skal slutte når en spiller når 3000 points				
Spillet skal let kunne oversættes til andre sprog				
Spillet skal let kunne skifte terningertyper				

Figur 5: Requirement traceability matrix

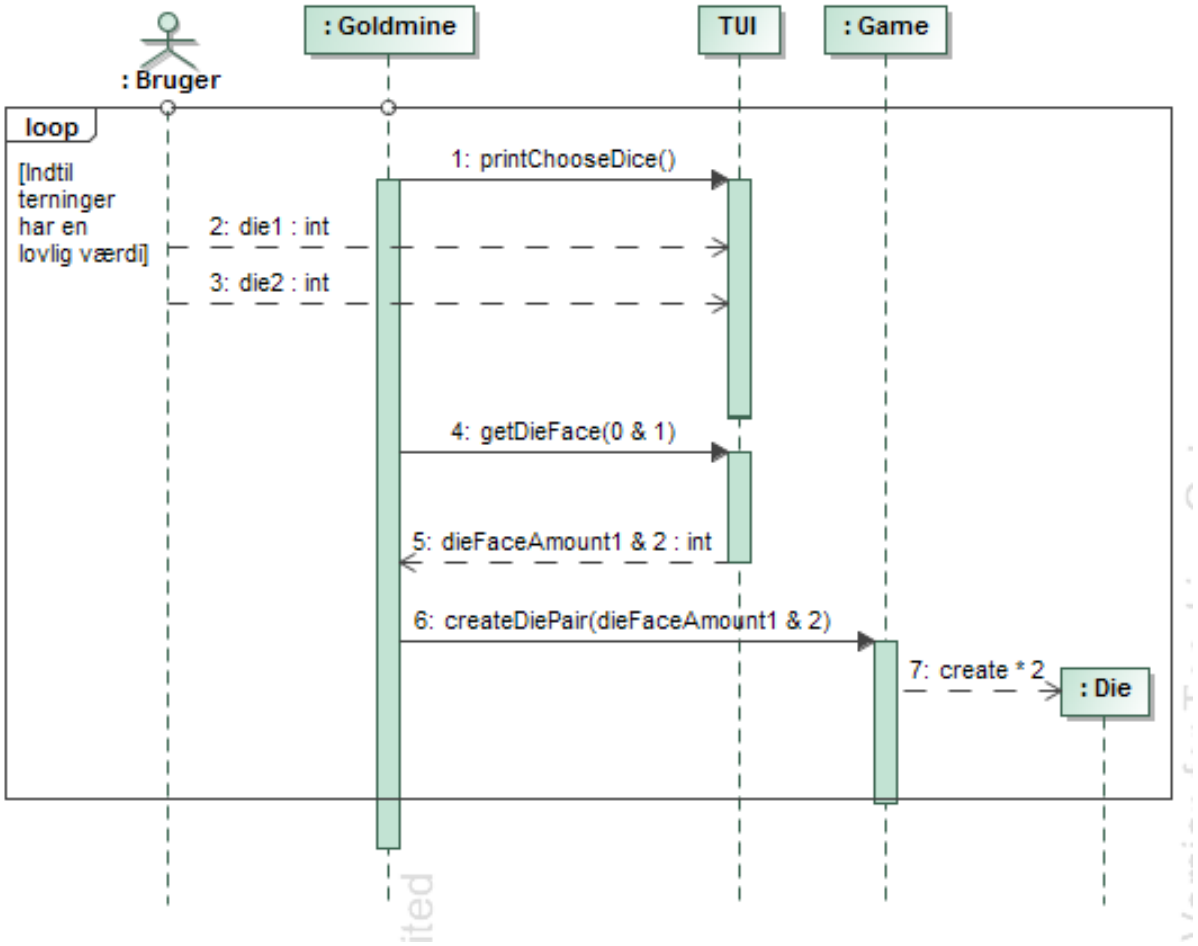
5 Bilag



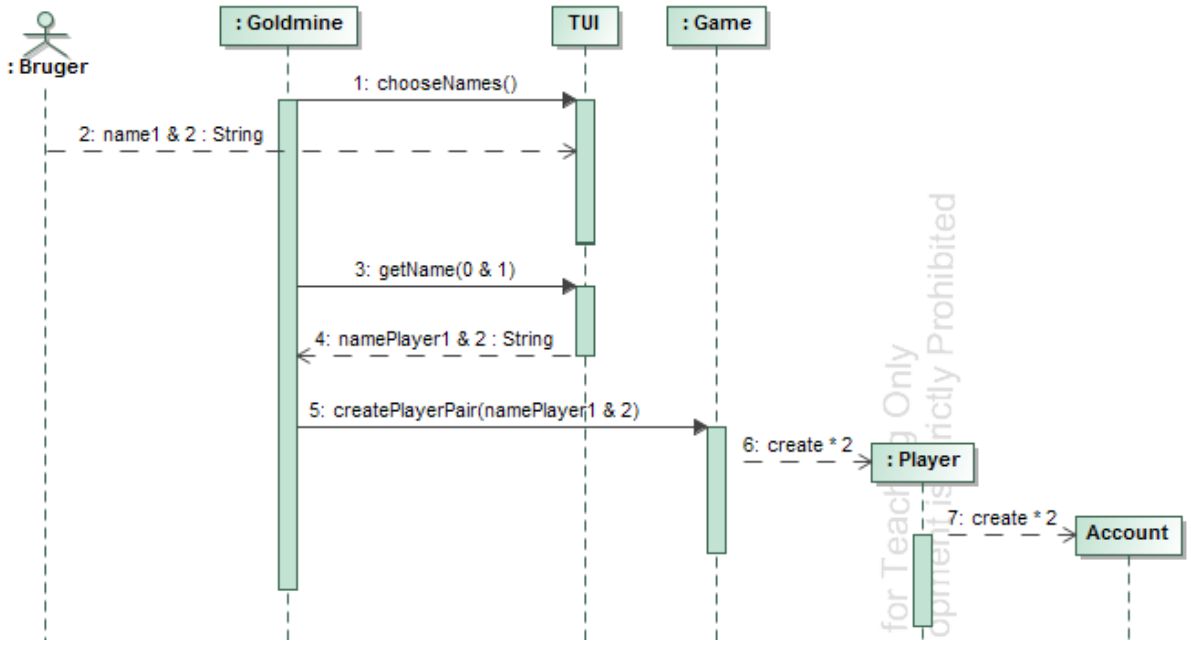
Figur 6: Det originale sekvensdiagram



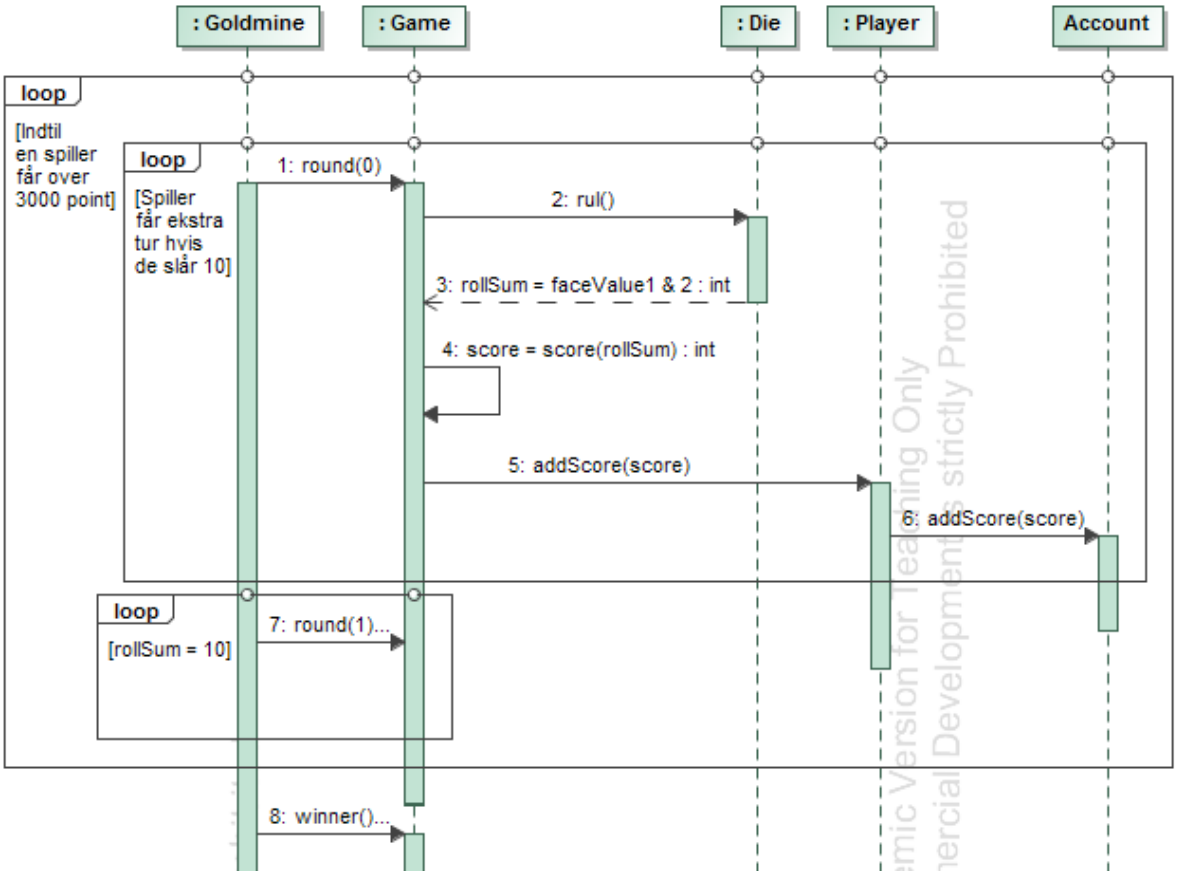
Figur 7: Sekvensdiagram over at vælge sprog



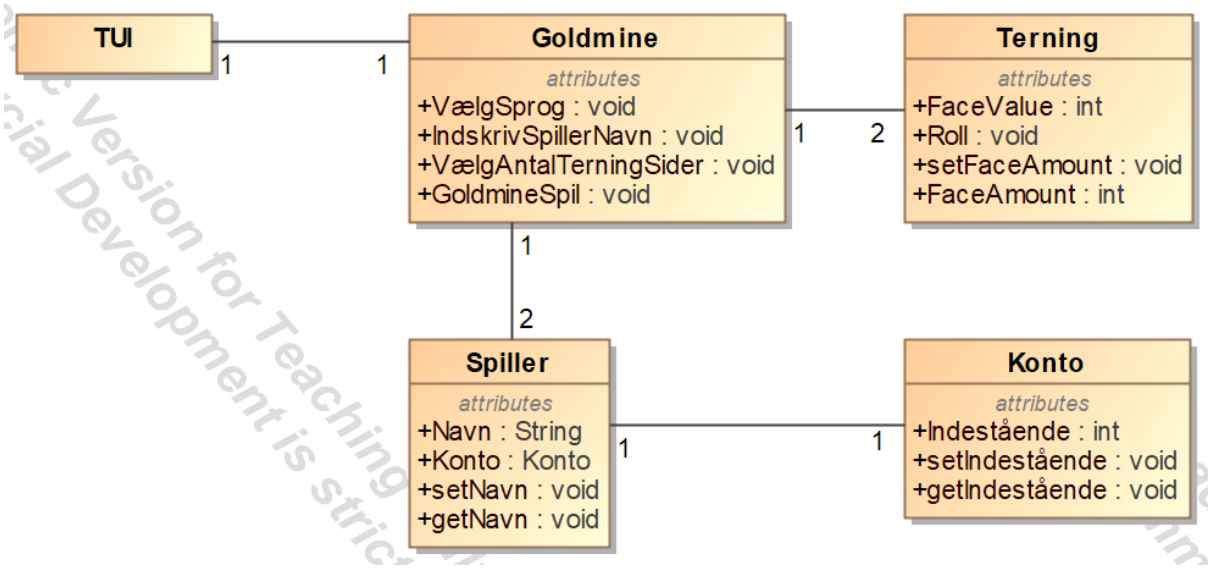
Figur 8: Sekvensdiagram over at vælge terninger



Figur 9: Sekvensdiagram over at vælge navn



Figur 10: Sekvensdiagram over spillets gang



Figur 11: Første design klassediagram

```
Indskriv spiller 1's navn:
Jonatan
Indskriv spiller 2's navn:
Søren
Jonatan's tur, tryk på Enter for at rulle terningerne.
```

Figur 12: Antal spillere i spillet

```
Du slog: 4 + 3 = 7
Du møder nogle monke. Selvom de ikke har meget, inviterer de dig til at hvile i deres kloster. (0)
Din score er nu: 1000
```

Figur 13: Start point

```
Du slog: 1 + 5 = 6
Du finder en by. Her sælger du lidt af de værdier du har fået samlet gennem din rejse. (+180)
Din score er nu: 2590

Jonatan's tur, tryk på Enter for at rulle terningerne.

Du slog: 3 + 1 = 4
På din rejse ser du et palads. Ejeren inviterer dig indenfor for at hvile og den næste dag giver han dig nogle forsyninger til din rejse (+100)
Din score er nu: 1680

Søren's tur, tryk på Enter for at rulle terningerne.

Du slog: 3 + 3 = 6
Du finder en by. Her sælger du lidt af de værdier du har fået samlet gennem din rejse. (+180)
Din score er nu: 2770

Jonatan's tur, tryk på Enter for at rulle terningerne.

Du slog: 4 + 4 = 8
Du udforkser en mørk hule. En bjørn hopper ud fra mørket og du løber! Du taber nogle forsyninger mens du løber væk. (-70)
Din score er nu: 1610

Søren's tur, tryk på Enter for at rulle terningerne.


Du slog: 1 + 1 = 2
Mens du går hen af en grus-sti ser du et tårn. Inde i tårnet finder du en kiste med 250 guld. (+250)
Din score er nu: 3020
```

Figur 14: 3000 point start, forskellige effekter og forskellige udprint

```
Jonatan's tur, tryk på Enter for at rulle terningerne.

Du slog: 6 + 4 = 10
Du har fundet vareulfvæggen. Om dagen er det en mand. Om natten er det en væg. (-80 + Ekstra tur!)
Din score er nu: 640

Jonatan's tur, tryk på Enter for at rulle terningerne.

Du slog: 4 + 1 = 5
Du falder om a udmattelse efter at have vandret gennem en kold ørken. Banditter røver dig mens du sover. (-20)
Din score er nu: 620

Søren's tur, tryk på Enter for at rulle terningerne.
```

Figur 15: Ekstra tur

```
Skriv antallet af ansigter på de to terninger, som spillet skal spilles med (samlede ansigter skal være 12 eller under!):
5
8
I har valgt terningerne: 5 & 8
Skriv antallet af ansigter på de to terninger, som spillet skal spilles med (samlede ansigter skal være 12 eller under!):
5
7
I har valgt terningerne: 5 & 7
Indskriv spiller 1's navn:
```

Figur 16: Terning skift

```
Vælg venligst et sprog (Please choose a language).
For at vælge dansk så klik 1 og Enter. To choose english press 2 and Enter.
2
You have chosen to play the game in english.
```

Figur 17: Oversætter