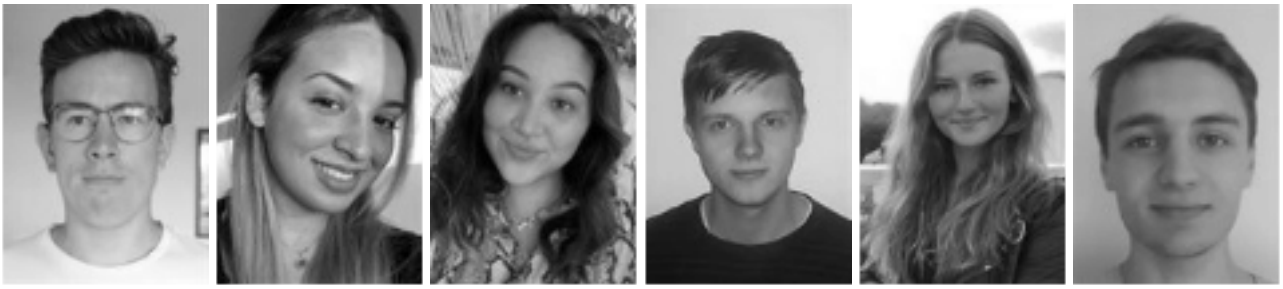
G15 CDIO delopgave 2

Indledende Programmering	Udviklingsmetoder til IT-Systemer	Versionsstyring og Testmetoder
02312-14	02313	02315

Afleveringsfrist: 1. november 2019

Gruppe 15



Christian	*Kamilia*	*Mina*	*Oliver*	*Sara*	*Tobias*
s184140	*s195466*	*s195381*	*s176352*	*s195388*	*s195458*

Denne rapport indeholder 20 Sider inkl. bilag og denne side.

Abstract (Kamilia)

This assignment is a combination of three courses, “Udviklingsmetoder til IT systemer”, “Indledende programmering” and “Versionsstyring og Testmetoder”. With the applications, MagicDraw, Github and IntelliJ, we have the ability of utilizing these to our main task, which is to produce a dice game which needs to fulfill the specific requirements given from a customer. This assignment includes various diagrams as well as the coding behind the dice game. Our aim is to create a new dice game that can be played by two people, where it’s possible to get either a negative or positive outcome depending on the field you land on. In the end it is concluded, that a proper functioning dice game can be programmed by using different components from the three courses.

Indholdsfortegnelse:

1. Indledning (Tobias) **4**
    1.1 Opgavebeskrivelse 4

2. Analyse **5**
    2.1 Krav (Mina, Tobias) 5
    2.2 Use cases (Sara) 7
    2.3 Domæne diagram (Mina og Christian) 8
    2.4 Systemsekvensdiagram (Kamilia og Sara) 9

3. Design **10**
    3.1 Klassediagram (Sara, Mina og Christian) 10
    3.2 Sekvensdiagram (Kamilia og Sara) 11

4. Implementering (Oliver & Tobias) **12**
    4.1 Klasser 12
    4.2 Metoder 13
        4.2.1 AccountModel 13
        4.2.2 PlayerModel 13
        4.2.3 GameModel 13

5. Test **14**
    5.1 Test af negativ balance (Christian) 14
    5.2 Test af Felt-værdier (Tobias) 14

6. Projektplanlægning **15**
    6.1 Tidsplan (Christian og Tobias) 15
    6.2 Projektforløb (Tobias og Sara) 16

7. Konfiguration (Oliver) **16**

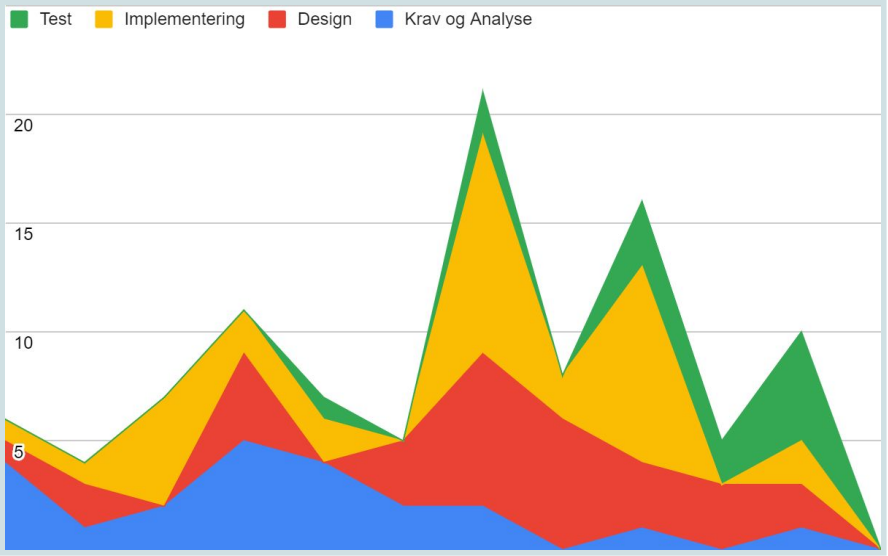
8. Konklusion (Sara) **18**

9. Bilag: **19**
    Bilag 1: Opgaveformulering 19
    Bilag 2: Felt tekster 22

Timeregnskab

dato	Krav og Analyse	Design	Implementering	Test
21/10/2019	4	1	1	0
22/10/2019	1	2	1	0
23/10/2019	2	0	5	0
24/10/2019	5	4	2	0
25/10/2019	4	0	2	1
26/10/2019	2	3	0	0
27/10/2019	2	7	10	2
28/10/2019	0	6	2	0
29/10/2019	1	3	9	3
30/10/2019	0	3	0	2
31/10/2019	1	2	2	5
1/11/2019	0	0	0	0
Timer i alt	**22**	**31**	**34**	**13**

Navn	Krav og Analyse	Design	Implementering	Test
Christian	2	4	11	5
Kamilia	5	4	2	0
Mina	4	9	2	0
Sara	6	6	2	0
Oliver	2	4	8	0
Tobias	4	4	11	8
Timer i alt	**23**	**31**	**36**	**13**



Graf over unified process for projektet

1. Indledning (Tobias)

Med succesen fra den tidligere opgave er vi blevet bedt om at lave et nyt spil der tager udgangspunkt i terningespillet. Programmet skal være et spil mellem 2 spillere igen, men hvor man lander på forskellige felter hver gang man slår med terningerne. Hvert felt skal have forskellige positive eller negative effekter for scoren.

Til udarbejdelse af produktet har vi taget brug af IntelliJ og GitHub og til diagrammer programmet Magicdraw. Disse programmer tilbyder muligheden for at alle kan arbejde projektet og at vi kan lave korrekte diagrammer der stemmer overens med koden.

Produktet er lavet med udgangspunkt i vores forskellige krav og designet ud fra vores domæne diagram. Hertil har vi så videre udviklet programmet ud fra vores use cases og med det udgangspunkt at dele af programmet skal bruges til videre projekter.

Vores repository til spillet kan findes her:
https://github.com/s184140/CDIO-2

1.1 Opgavebeskrivelse

I skal udarbejde et spil der bygger videre på opgaven fra CDIO 1. Spillet skal spilles mellem to spillere der slår med 2 terninger og hertil lander på numrene/felterne 2-12. Hvert felt har definerede positive eller negative konsekvenser som påvirker deres pengebeholdning der starter på 1000 ved spillets start. Den første spiller til 3000 vinder.

Projektet skal tage brug af terningerne fra projektet CDIO 1 og spilleren og hans pengebeholdning skal udarbejdes med tanke om at det kan bruges til efterfølgende projekter.

Udarbejdelsen af koden skal foregå så der er mulighed for at se udviklingen af projektet. Dette vil sige at man skal kunne se gruppemedlemmernes bidrag til projektet samt at der skal commits for hver gang de forskellige delopgaver er blevet løst eller forbedret.

2. Analyse

2.1 Krav (Mina, Tobias)

Kravliste

Kundens vision:

Kunden ønsker et system, som kan bruges på Windows maskinerne i databarerne på DTU. Spillet skal foregå mellem to spillere, og foregår ved at man som spiller slår med to terninger, som bestemmer hvilket felt man lander på. Hvert felt skal fremvise en tekst for feltets navn og dets effekt. Effekterne kas ses i Bilag 1, s.19. Resultatet skal herefter ændre spillerens totale sum af penge som starter på 1000 hvor første spiller der kommer op på 3000 vinder.

Det forventes at alle kan spille spillet uden videre vejledning. Faglige udtryk skal være naturlige, og der er ingen krav om koden og dokumentationen skal være på dansk eller engelsk. Hertil skal det også testes om det er muligt at få en negativ pengebeholdning for spillerne samt andre relevante test.

Kravspecifikation:

- Spillet skal kunne tilgås via maskinerne i DTU's databarer, uden bemærkelsesværdige forsinkelser.
- Spillet skal være mellem to personer
- Man skal kaste med to terninger ad gangen
- Turen skifter hver gang en spiller har kastet de to terninger
- Hver spiller starter med en pengebeholdning på 1000
- Spillet er slut når en spiller har fået en pengebeholdning på 3000
- Ved hvert kast kan man lande på et felt fra 2-12. Hvert felt har enten en positiv eller negativ effekt på spillerens pengebeholdning
- Ved hvert kast skal der udskrives en tekst angående det aktuelle felt spilleren lander på
- Spillet skal let kunne oversættes til andre sprog
- Det skal være let at skifte til andre terninger
- Spillernes data inkl. pengebeholdning skal kunne bruges i andre spil

- Der skal udføres en test som viser, at balancen aldrig kan blive negativ samt andre relevante tests. Kunden ønsker selv at kunne gentage disse tests.
- Der skal udarbejdes en beskrivelse af minimumskrav samt vejledning i hvordan kildekoden compiles, installeres og afvikles. Dette inkluderer en beskrivelse af hvordan koden importeres fra et git repository.

Funktionelle krav:

- Spillet skal være mellem to personer
- Man skal kaste med to terninger ad gangen
- Turen skifter hver gang en spiller har kastet de to terninger
- Hver spiller starter med en pengebeholdning på 1000
- Spillet er slut når en spiller har fået en pengebeholdning på 3000
- Ved hvert kast kan man lande på et felt fra 2-12. Hvert felt har enten en positiv eller negativ effekt på spillerens pengebeholdning
- Ved hvert kast skal der udskrives en tekst angående det aktuelle felt spilleren lander på
- Det skal være let at skifte til andre terninger

Non-funktionelle krav:

- Spillet skal kunne tilgås via maskinerne i DTU's databarer, uden bemærkelsesværdige forsinkelser.
- Spillet skal let kunne oversættes til andre sprog
- Spillernes data inkl. pengebeholdning skal kunne bruges i andre spil
- Der skal udføres en test som viser, at balancen aldrig kan blive negativ samt andre relevante tests. Kunden ønsker selv at kunne gentage disse tests.
- Der skal udarbejdes en beskrivelse af minimumskrav samt vejledning i hvordan kildekoden compiles, installeres og afvikles. Dette inkluderer en beskrivelse af hvordan koden importeres fra et git repository.

2.2 Use cases (Sara)

UC1 - Kast Terning:

- *Kaster terning indtil spillet er slut.*

figur 2.1.1: UC1 - Kast Terning | Diagram

Vores use cases beskrevet i en brief-udgave:

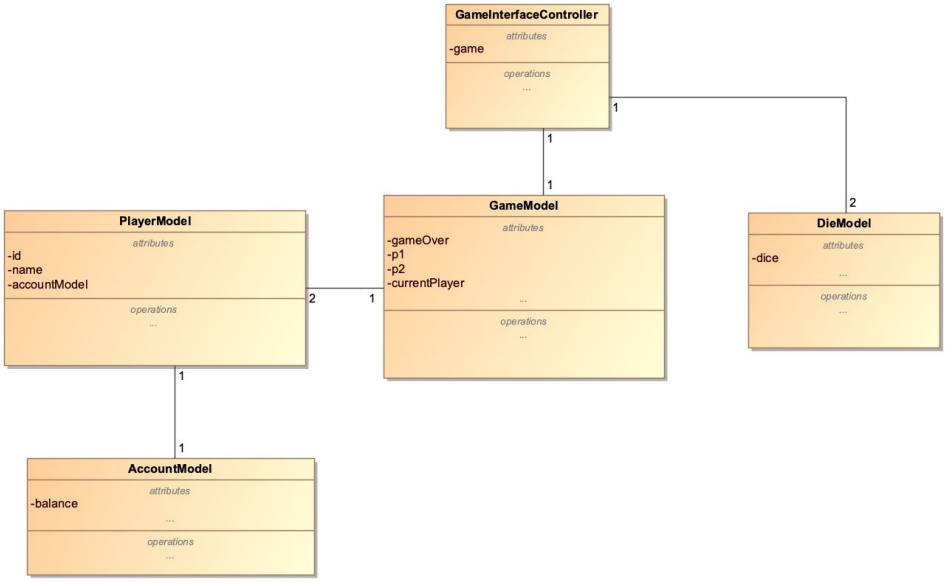
Use case:	*Beskrivelse*
Kast terninger	Spillerne skal kunne kaste med to terninger, og herefter se værdien af terningerne og scoren. Turen går herefter videre. Spillet er vundet ved 3000 point.

En central use case i en fully-udgave:

Use case: Kast terninger
UC: 1
Brief Description: Spiller kaster med to terninger
Primary Actors: Spillerne
Secondary Actors: Systemet
Preconditions: Spilleren er inde i spillet. Spilleren har indtastet navnene.

Main flow: 1. **Spilleren kaster med terningerne** 2. **Spilleren ser sin ternings værdi og score.** 3. **Turen går videre til næste spiller** 4. **Spillet er vundet når en spiller har 3000 point.**
Postconditions: Spilleren har kastet med terningerne og scoren er ændret herefter. Næste spiller skal kaste med terningerne. Når scoren op på 3000, har spilleren vundet.

2.3 Domæne diagram (Mina og Christian)



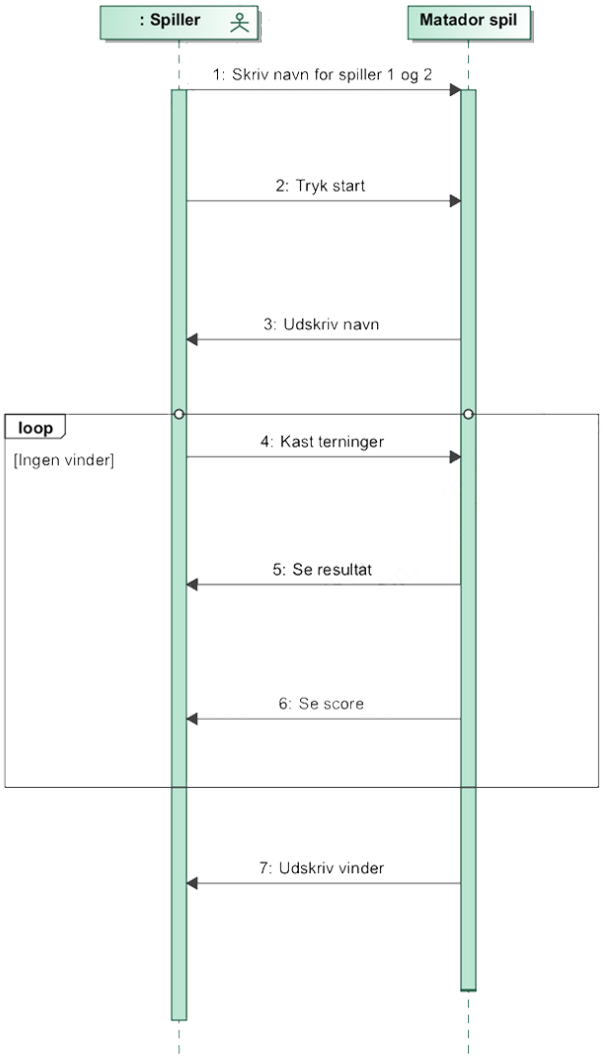
Figur 2.3.1 - Domæne diagram

Vores domæne diagram består af klasserne:

- GameInterFaceController
- AccountModel
- GameModel
- DieModel
- PlayerModel

Domæne diagrammet forklarer relationer mellem de forskellige elementer i vores system. I et dagligdags sprog alle kan forstå.

2.4 Systemsekvensdiagram (Kamilia og Sara)

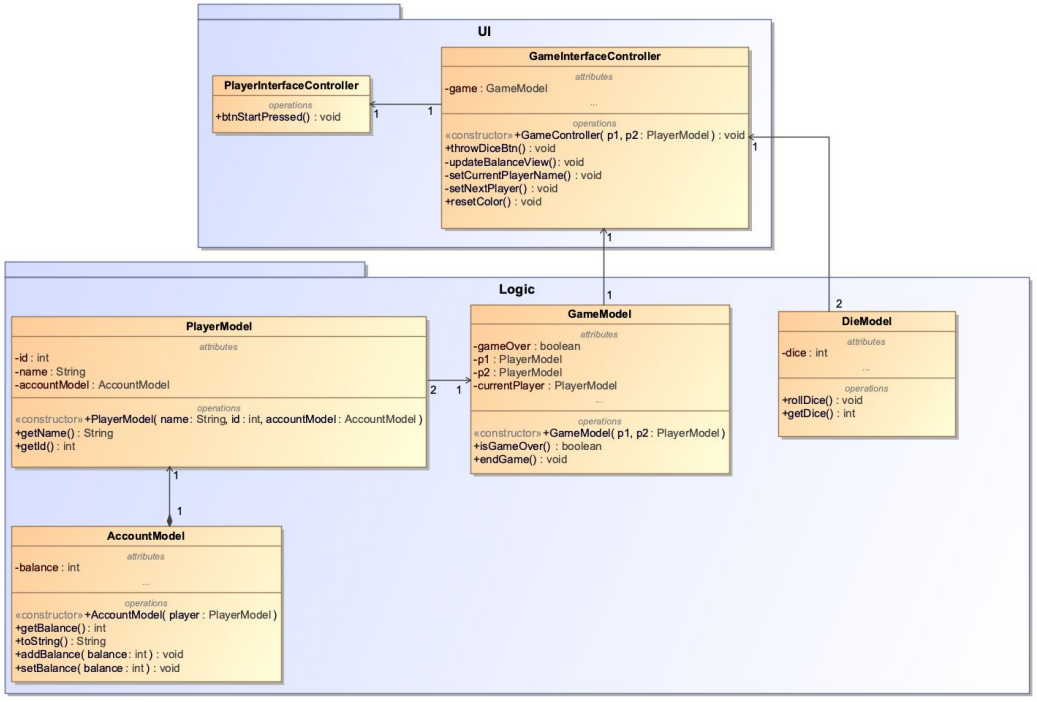


Da vores use cases er meget simpel, har vi valgt at lave vores systemsekvensdiagram over hele spillet. Vores terningespil kører mellem en spiller og spillet. Til at starte med, skal spillerne skrive sit navn, og derefter trykke på start. Herefter returnere spillet navnene. Spillerne skal kaste terninger, og derefter returnere spillet resultatet og den samlede score. Det kan af fase 4-6 ses, at hvis ikke der er defineret en grænse for en vinder, kører disse i et loop. Til sidst udskriver spillet en vinder.

Figur :Systemsekvensdiagram

3. Design

3.1 Klassediagram (Sara, Mina og Christian)



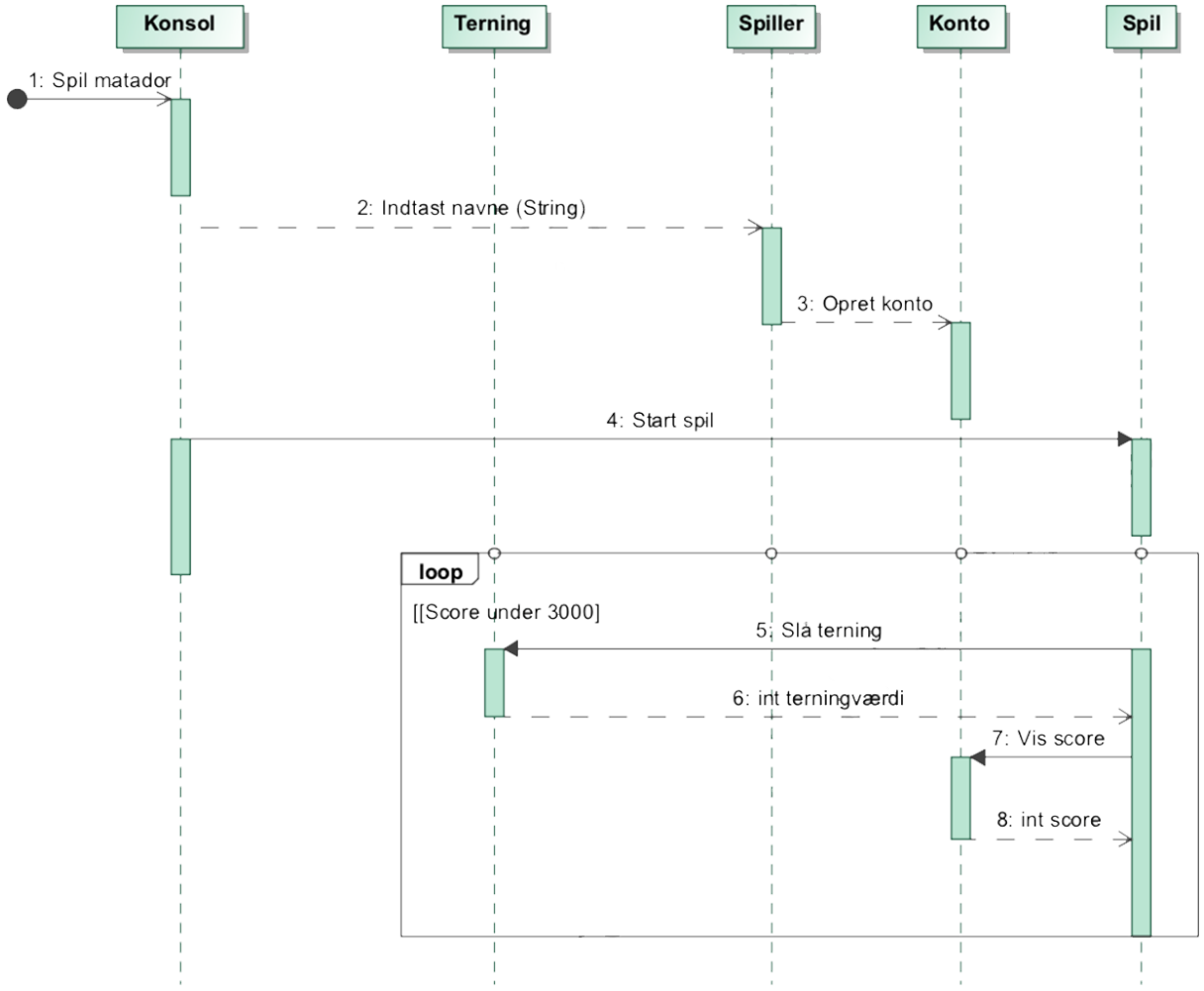
Figur 3.1.1 - Klasse diagram

Vores klassediagram består af klasserne:

- PlayerInterfaceController
- GameInterfaceController
- PlayerModel
- AccountModel
- GameModel
- DieModel

Samt af pakkerne "UI" og "Logic". Pakken UI består af klasserne med koden der interagerer med brugeren. Såsom start-spil (btnStartPressed), kast terning(throwDiceBtn), opdater spillertur (setNextPlayer) og opdater balance (updateBalanceView). Pakken Logic indeholder spil logikken. Såsom indlæsning af de to spillere, gennem en constructor PlayerModel, som benytter inputtet fra UI'et. Derudover den logiske konstruktion terningen i DieModel, hvis værdi indlæses i GameInterfaceController, som udløser en given balance-ændring, som behandles i AccontModel.

3.2 Sekvensdiagram (Kamilia og Sara)



Figur :Sekvensdiagram

Vi har samlet vores sekvensdiagram over hele vores projekt, da det ikke er så omfattende. Her kan man se forbindelserne mellem konsollen, terningerne, spillerne, kontoen og selve spillet. Igen har vi et loop, men denne gang opløftet af vores grænse på 3000 point, hvor en spiller har vundet.

4. Implementering (Oliver & Tobias)

Da projektet er en videreudvikling fra CDIO-1, minder vores kode meget om det tidligere projekt. Som vi har beskrevet i problemformuleringen, så er spiller- og terningeklassen indsat fra CDIO-1, men frem for strukturen i det første projekt, så har vi brugt MVC princippet i opbyggelsen af koden. Vi har valgt at benytte JavaFX til GUI delen af projektet, da JavaFX er simpelt at implementere og benytte.

4.1 Klasser

Koden er udarbejdet ud fra Model-View-Controller, MVC, princippet.
Vi har to controllers, GameInterfaceController og PlayerInterfaceController, som styrer, når brugeren/spilleren laver en interaktion med henholdsvis gameInterfaceView og playerInterfaceView. Controllerne kan så hente og opdatere data fra de modeller som controlleren har behov for.
Således har vi opdelt vores projekt i to forskellige pakker.

UI (package)
- gameinterface (package)
  - GameInterfaceController.java
  - gameInterfaceView.fxml
- playerinterface (package)
  - PlayerInterfaceController.java
  - playerInterfaceView.fxml

Logic (package)
- PlayerModel.java
- DieModel.java
- AccountModel.java
- GameModel.java

I programmet er der også en Main klasse, hvis eneste funktion er at initialisere og vise PlayerInterfaceView. I PlayerInterfaceView kan man se at dens controller er defineret "`fx:controller="sample.ui.playerinterface.PlayerInterfaceController""`. Dette gør at brugerhandlinger i viewet, kan være koblet til en metode, eller at en komponent kan være koblet til et attribut i controlleren.

Når man har klikket "Start spil" i viewet bliver GameInterfaceView initialiseret og vist. Den har så en reference til controlleren GameInterfaceController. Modellerne bliver opdateret når

der foregår en ændring. For eksempel når en spiller "kaster med terningen" og får nogle point.

4.2 Metoder

4.2.1 AccountModel

AccountModel er simpel da dens eneste funktioner get-, add- og setBalance kun ændrer og returnerer spillerens balance. For denne klasse bliver toString metoden også overskrevet. toString metoden returnerer balancen konverteret til en String.

```
private int balance = 1000;

public int getBalance() { return this.balance; }
public void addBalance(int balance){...}
public void setBalance(int balance){...}

//Man kunne også bare kalde denne getBalanceToString da den ikke ret
@Override
public String toString() { return Integer.toString(this.balance); }
```

4.2.2 PlayerModel

Her defineres et id og navn for spilleren. En konto bliver også defineret i klassen. Når man initialiserer PlayerModel klassen så skal man sende et id, et navn og et objekt af en AccountModellen med. Metoderne er meget simple og returnerer diverse attributters indhold.

```
private int id;
private String name;

private AccountModel accountModel;

public PlayerModel(int id, String name, AccountModel accountModel){...}

public AccountModel getAccountModel() { return accountModel; }

public String getName() { return this.name; }
public int getId(){ return this.id; }
```

4.2.3 GameModel

GameModel holder "øje" med selve spillet. Den indeholder attributter af de to spillere som indgår og en attribut til at bestemme den aktuelle spiller. Sidste attribut er en boolean, gameOver, som bliver sat til falsk når en vinder er fundet. Metoderne er også her ret så simpel.

```
private boolean gameOver;

private PlayerModel p1, p2;
private PlayerModel currentPlayer;

public GameModel(PlayerModel p1, PlayerModel p2){
    this.gameOver = false;
    this.p1 = p1;
    this.p2 = p2;
}

public PlayerModel getPlayer1() { return p1; }

public PlayerModel getPlayer2() { return p2; }

public PlayerModel getCurrentPlayer() { return currentPlayer; }

public void setCurrentPlayer(PlayerModel player) { this.currentPlayer = player; }

public boolean isGameOver() { return gameOver; }
public void endGame() { gameOver = true; }
```

5. Test

For at se om programmet kører ordentligt har vi lavet nogle tests. Den første tjekker om der er mulighed for at kunne få en negativ balance. Den anden test ser på de givne felt værdier vi har fået fra kundens vision og ser om de giver mening og ville virke praktisk når man spiller spillet.

5.1 Test af negativ balance (Christian)

Vi er blevet bedt om at teste for at balance ikke kan blive negativ. Til dette har vi lavet en separat branch kaldet “account_test” hvor vi så kører spillet med nogle ændrede værdier i switch-case udfaldene. Alle de positive penge tilførsler blev tilført et minus, så spilleren kun mistede penge. Det fremstod efter et vis antal klik, at begge spiller forblev på en fast penge-balance på 0. Vi havde nemlig opbygget koden så hver gang man tilføjer noget til ens balance bliver der tjekket for om balancen er negativ og sætter den til 0 i det tilfælde.

5.2 Test af Felt-værdier (Tobias)

Vi ville gerne se om spillet var retfærdigt og om det havde nogle fornuftige værdier for at spillet skulle være sjovt og interessant eller om det overhovedet var muligt at gennemføre.

Derfor har vi lagt alle felternes effekter sammen for at se om de ville give en gennemsnitlig positiv værdi. Dette kan gøres ud fra 36 kast da alle resultater har en chance fra 1-6 ud af 36

Resultat	2	3	4	5	6	7	8	9	10	11	12
Chance	$\frac{1}{36}$	$\frac{2}{36}$	$\frac{3}{36}$	$\frac{4}{36}$	$\frac{5}{36}$	$\frac{6}{36}$	$\frac{5}{36}$	$\frac{4}{36}$	$\frac{3}{36}$	$\frac{2}{36}$	$\frac{1}{36}$
Effekt	+ 250	−100	+ 100	−20	+ 180	0	−70	+ 60	−80	−50	+ 650

Gennemsnitligt Total efter 36 kast

$$((250+650)\cdot 1)+((-100-50)\cdot 2)+((100-80)\cdot 3)+((-20+60)\cdot 4)+((180-70)\cdot 5)+((0)\cdot 6)$$

1370

Gennemsnitlig ændring per kast

$$\frac{1370}{36}=\frac{685}{18}=38.056$$

Gennemsnitlig antal kast for at vinde

$$\frac{2000}{38.056}=52.55413075$$

Vi kan se at det gennemsnitlige resultat efter 36 kast er en stigning på 1370 og en gennemsnitlig stigning per kast på *ca*.38. Med dette kan vi dog også udregne hvor mange kast det vil tage generelt at vinde spillet. Da man starter på 1000 og skal ende på 3000 skal man opnå 2000 hvilket gennemsnitligt ville kræve 52½ kast, men da man ikke kan lave et halvt kast er det 53. Mulige ændringer eller forbedringer kunne være at ændre værdierne på felterne eller slut målet for at spillet skulle gå hurtigere eller langsommere

6. Projektplanlægning

6.1 Tidsplan (Christian og Tobias)

Forinden projektets start lavede vi en tidsplan med udgangspunkt i UP. Hertil kan det ses på vores timeregnskab og beskrevet i projektforløbet hvordan projektet er gået og hvor meget vi har holdt os op af tidsplanen.

Aktivitet	Underaktivitet	ansvarlige	21/10	22/10	23/10	24/10	25/10	26/10	27/10	28/10	29/10	30/10	31/10	1/11
Indledning	Abstract													
	Indledning													
Krav og analyse	Kravliste													
	Use Case diagram													
	Domæne diagram													
	Systemsekvensdiagram													
Design	Designklassediagram													
	Sekvensdiagram													
	GRASP-mønstre													
Implementering	Kode eksempel													
Test	Balance ikke negativ													
	Andre test													
Konklusion														
Kode skrivning	GUI													
	Klasser													
	Test													
	UI													
	Konfiguration													

6.2 Projektforløb (Tobias og Sara)

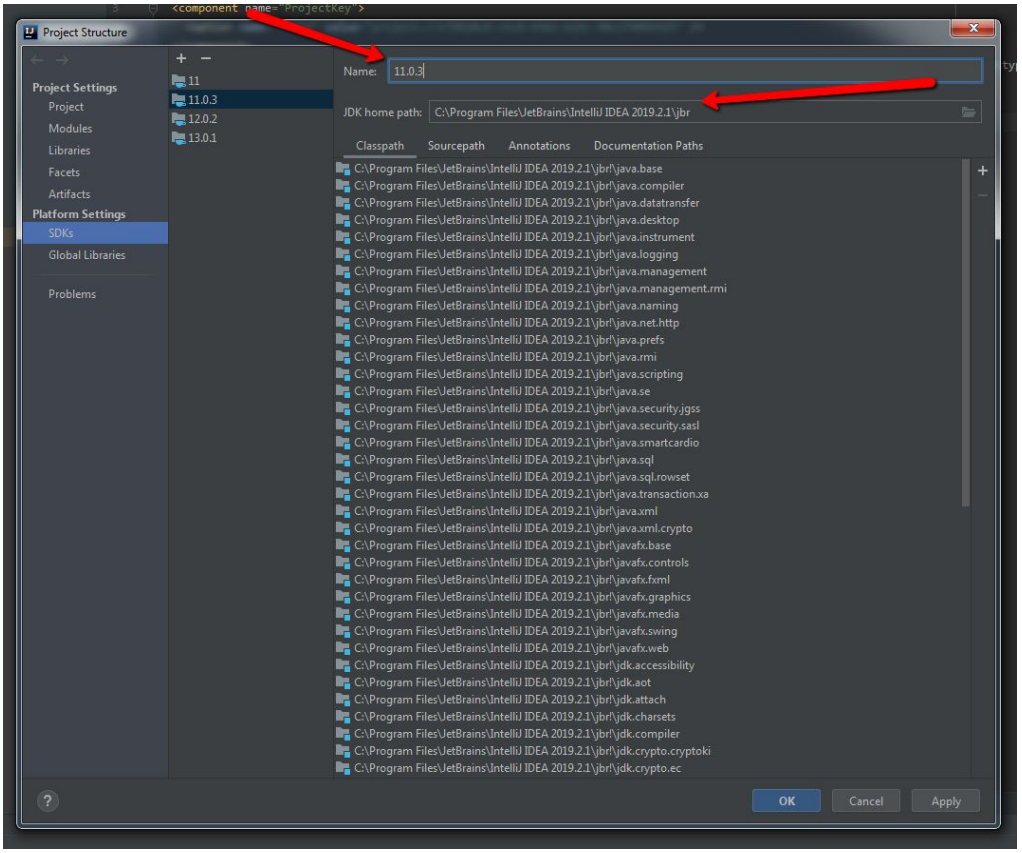
Vi planlagde vores projekt dagen for udleveringen af opgavebeskrivelsen i ugen op til efterårsferien. Da nogle af gruppemedlemmerne havde en eksamen at læse op til i ferien og andre havde opgaver til efter ferien aftalte gruppen først at begynde hoveddelen og videre planlægning efter ferien. Dog aftalte vi at læse op på projektet og lave forberedende noter til rapporten og produktet til nå vi skulle i gang. Vi startede ud på ugen med at mødes og snakke om hvordan vi ville gribe projektet an. Vi aftalte her at dele projektet op til de forskellige gruppemedlemmer hvor hvert gruppemedlem var ansvarlig for et emne sammen med mindst en anden fra gruppen. Dette var for at fordele arbejdet mest muligt, samt for at sørge for at alle havde gjort noget ved næsten alle dele af rapporten. Vi sørgede hertil også for at planlægge arbejdet så alle var involverede i programmeringsdelen af produktet og at alle lavede en tilføjelse. Vi planlagde alle at mødes om søndagen før aflevering af opgaven, hvilket fungerede rigtig godt. Her arbejdede alle på opgaven, og vi nåede sammen rigtig langt. Vi manglede om tirsdagen at skrive om GRASP mønstre, hvilket vi gerne ville have hjælp til af vores hjælpelærer, og derfor kunne vi efter timen tirsdag færdiggøre vores arbejde.

7. Konfiguration (Oliver)

I projektet har vi benyttet os af følgende

- Java JDK 11.0.3 med Java FX
- InteliJ
- Scenebuilder

Ved start af Intelij vælger man “Check out from Version Control” og vælger herunder git.
I URL indtastet “https://github.com/s184140/CDIO-2” for efterfølgende at klikke Clone. Når projektet er indlæst vil der muligvis komme en fejl med at SDK’en ikke er specificeret. For at løse dette vælges File -> Project Structure. Inde i denne menu vælges SDKs. Herinde klikkes på + og så JDK for så at navigere frem til “C:\Program Files\JetBrains\IntelliJ IDEA 2019.2.1\jbr” eller der hvor ens JBR ligger. Når denne er valgt er vigtigt at man navngiver denne til 11.0.3
Se billedet nedenfor:



Programmet skulle nu kunne køres ved at klikke på den grønne pil.

Hvis man ønsker at redigere i view filerne, .fxml filerne, og gerne vil have et redigeringsprogram, så hentes SceneBuilder. .fxml filerne åbnes separet med SceneBuilder.

OBS. Det er til at bemærke, at der kommer en WARNING i terminalen når projektet køres. Dette kan man på nuværende tidspunkt blot se bort fra, da det er en fejl i InteliJ, når man benytter sig af SceneBuilder uden for InteliJ. Selve advarslen forårsager ikke fejl i selve programmet.

8. Konklusion (Sara)

Ud fra arbejde med metoder fra de tre kurser har vi har udarbejdet en fuldendt rapport, hvor vi har taget forbehold for misforståelser, som kan opstå under implementering af et projekt. Vi har igen oprettet en GitHub gruppe, så alle i virksomheden kan versionere og holde sig opdateret med opbygningen af vores kode, som vi er blevet færdige med. De mål vi havde for opgaven ved start, er vi nået i mål med, og vi opfylder dermed kundens krav, som vi har forstået dem. Vi kan konkludere, at vi har fået oprettet endnu en GitHub gruppe, en rapport som uddyber kundens forespørgsler ved brug af kravspec., use cases og forskellige modeller og at vi har fået kodet spillet som kunden forespurgte.

Samarbejdet i gruppen er gået godt. Vi brugte mere tid denne gang på at mødes, så vi bedre kunne følge med i hvad hinanden lavede. Ellers overholdt gruppen sine aftaler, som vi lavede hjemme. Igen har vi hjulpet hinanden hvor vi kunne, og vi har gjort hvad vi kunne for, at alle forstod den endelige rapport. I vores videre arbejde, vil vi igen bytte rundt på rollerne i vores projektopgave skrivning, så alle får prøvet lidt af hvert, og dermed en god forståelse for det hele.

9. Bilag:

Bilag 1: Felt tekster

2	Tower +250	“Du reddede prinsessen i tårnet, og kongen giver dig 250 mønter!”
3	Crater -100	“Du faldt ned i et krater, og tabte 100 mønter derned”
4	Palace gates +100	“Portene til Paladset åbner for dig, og som velkomst får du 100 mønter”
5	Cold Desert -20	“Du er faret vildt i den kolde ørken, og bruger 20 mønter på at holde varmen”
6	Walled city + 180	“Du finder sikkerhed i en by omringet af mure, +180”
7	Monastery 0	“Du overnatter i et kloster, +0”
8	Black cave -70	“Du falder ned i en sort grotte, -70”
9	Huts in the mountain +60	“Du finder ly i en hytte i bjergene, +60”
10	The Werewall (werewolf-wall) -80, men spilleren får ekstra tur	“Du bliver overfaldet af ulve-muren, de stjæler 80 mønter men du får en ekstra tur.
11	The pit -50	“Du bliver smidt i “hullet” og mister samtidigt 50 mønter”
12	Goldmine +650	“Du fandt en guldmine og får 650 mønter!”