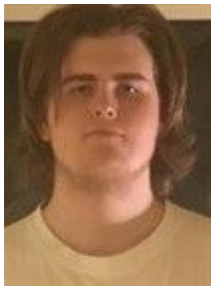
CDIO Del 3

(02313) Udviklingsmetoder til IT-systemer
(02314) Indledende programmering
(02315) Versionsstyring og testmetoder

Gruppe nr.: 39



s195828
Benjamin Andersen

s141479
Anthony Haidari

s163053
Anton Ø. Schmidt

s176481
Reza Haddang

s195484
Sid Ali Mounib

s180646
Anders Christensen

Indholdsfortegnelse:

Indledning

For dette projekt har vi lavet et program, og en rapport omkring programmet. I rapporten kan man se kundens vision for spillet, og de krav og ønsker som kunden har til os. Man kan også se alle de ting som vi har inklueret i spillet som felterne, kontiene osv.
Vi har også lavet en analyse hvilket indkluder flere diagrammer, som viser hvad programmet gør når man bruger det, og hvordan programmet virker. Det har vi gjort ved at forklar de metoder og objekter vi anvender til at lave spillet. Vi har også beskrevet hvad det kræver for at kunne køre programmet, og hvilke værktøjer vi har brugt til at koordinere arbejdet.
Derudover har vi også lavet 3 testcases, som er med til at bevise at det endelige spil fungere som det skal og overholder kravene og ønskerne fra kunden.

Krav

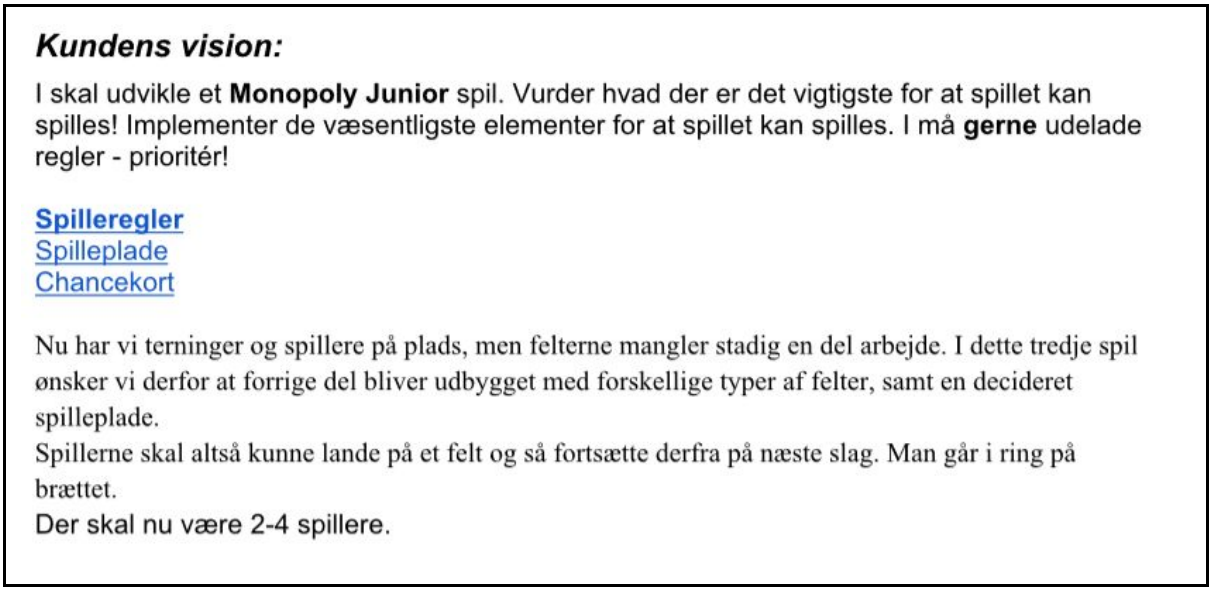
Generelt defineres krav som et sæt af betingelser som det pågældende system, eller i mere bredt forstand projektet der skal udvikles på, skal være i overensstemmelse med kundens ønsker.
Det betyder, at vi som en organisation skal i samarbejde med kunden have en systematisk tilgang til at finde, dokumentere, organisere og spore de ændrede krav i systemet.
Hertil er ordet "finde" meget vigtig, hvorfor vi opfordrer kunden til at samarbejde og via teknikker som skrivning af use cases med kunden, workshops, en demo af resultaterne af hver iteration til kunden og løbende modtage feedback fra kunden.

Kravliste er udarbejdet og beskrevet under funktionelle/ikke funktionelle krav.
Vi benytter os af UP (Unified Process), og vil gennemgå vores planlægning, fremgangsmåde, samt udvikling løbende, og vil derfor ikke nødvendigvis implementere alle nedenstående punkter, hvis de tidsmæssige ressourcer er for begrænset.

Et krav-liste er udarbejdet ud fra spillets regler og dette kan forefindes i sektionen nedenfor, under "kravspecifikation".
De systemmæssige krav for eksekvering af programmet, vil kunne findes under sektionen "konfigurationsstyring".

Kundens version

Kunden ønsker udvikling af brætspillet "Matador Junior", og har givet følgende specifikationer til projektet:



Kundens vision:

I skal udvikle et **Monopoly Junior** spil. Vurder hvad der er det vigtigste for at spillet kan spilles! Implementer de væsentligste elementer for at spillet kan spilles. I må **gerne** udelade regler - prioritér!

Spilleregler
Spilleplade
Chancekort

Nu har vi terninger og spillere på plads, men felterne mangler stadig en del arbejde. I dette tredje spil ønsker vi derfor at forrige del bliver udbygget med forskellige typer af felter, samt en decideret spilleplade.
Spillerne skal altså kunne lande på et felt og så fortsætte derfra på næste slag. Man går i ring på brættet.
Der skal nu være 2-4 spillere.

Billede 1: Kundens vision

Der har dermed været behov fra udviklerne til at uddybe kravspecifikationer, ud fra spillets regler, samt skabe formulering for andre fornødne krav for implementering, da kundens ønsker er meget overordnet.
Efter konsultation med underviser og hjælpelærer, er der blevet udarbejdet kravliste, som er beskrevet nedenfor, under "Funktionelle Krav",

Funktionelle krav / Kravliste

- Skal kunne spilles af 2-4 spiller
- Systemet/spillet varetager rollen som "Bankør" hvor valutaen er Monopoly penge og angives med et stort M og beløbet bagefter ex. 20 monopoly penge skrives: M20
    - For aktiv 2 spiller gives startbeløb på: M20 hver
    - For aktiv 3 spiller gives startbeløb på: M18 hver
    - For aktiv 4 spiller gives startbeløb på: M16 hver
- En spiller kan højest eje 12 felter
- Alle spiller burde kunne se likvid beholdninger.
- Systemet skal varetage de automatiseret dele i spillet, såsom betaling af ejendomskøb/husleje/afgift, når spiller lander på felt med sådanne krav.
- Spillet skal kunne gemmes.
- 20 chancekort.
    - 4 chancekort er specifikt rettet mod spiller brik
    - 7 chancekort er specifikt rettet mod bræt felter opdelt i farver

    - 1 chancekort er specifikt lavet til at gemmes til næste gang spilleren ryger i "fængslet" hvor spiller så undgår at betale M1 for at komme ud næste gang.
    - 2 chancekort er specifikt lavet til at rykke spiller frem til et bestemt felt (feltet: skateparken og strandpromenaden)
    - 1 chancekort er specifikt lavet til at rykke spiller frem til start
    - 1 chancekort er specifikt lavet til at rykke 3 felter frem.
    - 1 chancekort er specifikt lavet til at rykke 1 felt frem eller tage et nyt chancekort
    - 1 chancekort er specifikt lavet til at straffe dig hvor du skal betale M2
    - 1 chancekort er specifikt lavet til at du modtaget M1 fra de andre spiller (du modtager M1-3 alt efter antallet af spiller ud over ham som trak kortet. ud over personen som trak kortet mister alle spiller M1)
    - 1 chancekort er specifikt lavet til at du modtaget M2

- Der benyttes af en digital terning, med værdierne 1-6, som angiver hvor mange felter hver spiller skal bevæge sig med uret. Den yngste spiller starter, og skifter dernæst tur, indtil spil afsluttes.

- Spil afsluttes når:
  - Spiller ikke har likvid beholdning til betaling af: køb af ejendom, husleje, eller afgifter fx chancekort eller fængslet:
  - Spiller med flest penge vinder, når spillet afsluttes.

- Brættes felter:
  - Spiller vil bevæge sig med uret, ud fra terningkast værdi
  - Spiller lander på ledige felter: Køb til pris
  - Spiller lander på felt ejet af samme spiller på felt: Intet foretages
  - Spiller lander på felt ejet af en anden spiller: betal husleje
  - Dobbelt felter, ejet af samme spiller, resulterer i dobbelt husleje.
  - hver gang en spiller lander på eller bevæger sig forbi start feltet belønnes spilleren med M2
  - Chance Felt, giver spiller et vilkårlig chance kort ud fra de 20 givet.
  - Gå i fængsel felt: Pågældende spiller fængsels. Spiler passerer ikke feltet "START", og modtager derfor hellere ikke M2. For løsladelse, betales der M1 af pengebeholdning, eller brug af chancekortet "Du løslades uden omkostninger". Når løsladt, forsættes kast med terning som normalt. Spiller modtager stadig husleje, mens vedkommende er i fængsel.
  - På besøg: Foretag intet, spiller er blot på besøg
  - Gratis Parkering: Foretag intet, spiller har gratis parkering.

Feltnummer	Feltnavn	Pris	Beskrivelse
1	Start	+ M2	Spiller modtager M2 for hvert passering af felt
2	Burgerbaren	- M1	Spiller køber ejendom el. betaler husleje
3	Pizzaria	- M1	Spiller køber ejendom el. betaler husleje
4	Chance		Spiller modtager chancekort
5	Slikbutikken	- M1	Spiller køber ejendom el. betaler husleje
6	Iskiosken	- M1	Spiller køber ejendom el. betaler husleje
7	Fængsel		Spiller på besøg: foretag intet, spiller er på besøg.
			Spiller i fængsel: Betal M1, el. benyt chancekort.
8	Museet	-M2	Spiller køber ejendom el. betaler husleje
9	Biblioteket	-M2	Spiller køber ejendom el. betaler husleje
10	Chance		Spiller modtager chancekort
11	Skaterparken	-M2	Spiller køber ejendom el. betaler husleje
12	Swimmingpoolen	- M2	Spiller køber ejendom el. betaler husleje
13	Parkering		Foretag intet. Spiller får gratis parkering
14	Spillehallen	-M3	Spiller køber ejendom el. betaler husleje
15	Biografen	-M3	Spiller køber ejendom el. betaler husleje
16	Chance		Spiller modtager chancekort
17	Legetøjsbutikken	-M3	Spiller køber ejendom el. betaler husleje
18	Dyrehandlen	-M3	Spiller køber ejendom el. betaler husleje
19	Fængsel		Spiller flyttes til feltet "Fængsel"
20	Bowlinghallen	-M4	Spiller køber ejendom el. betaler husleje
21	Zoo	-M4	Spiller køber ejendom el. betaler husleje
22	Chance		Spiller modtager chancekort
23	Vandlandet	-M5	Spiller køber ejendom el. betaler husleje
24	Strandpromenaden	-M5	Spiller køber ejendom el. betaler husleje

Table 1: Beskrivelse af brættets felter

Non-funktionelle krav / kvalitetskrav

- Sikkerhed - spillerne skal ikke kunne snyde og ændre deres likviditet eller hvilket felter der er i deres besiddelse. samt heller ikke kunne snyde ved at ændre terningens “facevalue” eller max/min værdierne. eller flytte sin brik fra et felt til et andet eller flytte positionen af felter om
- Pålideligt - vi skal være sikker på at vores terning math random er random ved hjælp af en test kan det testes hvis terningerne slås 600 slag skal ca 100 slag laden på værd af de mulige øjne som terningen viser fra 1-6 men en afvigelse på 5% acceptabel. det samme kan siges for chancekortene de skal også være tilfældigt og det skal testes med en afvigelse på under 5%
- Responstid - Programmet/koden skal kunne køres uden unødvendige forsinkelser, sådan at bruger/spilleren får en god oplevelse af spillet. Vi foreslår et responstid på 1 sekunder.
- Brugervenlighed - Spillerne skal kunne tilgå spillet uden brug af vejledning.
- Reglementerne for at kunne forstå spillet samt dets formål skal være dokumenteret.

Analyse

Analyse går i al sin enkelthed ud på at lave en undersøgelse af et softwaremæssigt problemområde hvor der især lægges vægt på at undersøge de specifikationer og krav softwaren skal opfylde. Via analysen forsøges der snarere på, at finde svar på hvordan det kommende software eller IT-system bruges af omgivelserne og hvilke funktioner systemet skal stille til rådighed samt grænsefladerne, end at finde en reel løsning.

I løbet af det følgende objektorienteret analyse, forsøges der derfor på at finde og beskrive objekter der kan være behjælpelig med at designe, udvikle samt implementere Matador Junior spillet. Hertil anvender vi diverse konceptuelle udviklingsmetoder og værktøjer, såsom kravanalyse, use case analyse og UML-diagrammer. Men det er på sin plads at først analysere de mennesker og organisationer der får hel eller delvis indflydelse på det nærværende projekt og på den måde bliver i en vis grad også påvirket af det. Vi starter derfor med at lave en interessentanalyse.

Interessentanalyse

En interessent kan defineres som en, der påvirker projektet eller en der bliver påvirket af projektet. Interessentanalysen er et stærkt værktøj, der hjælper os med at skabe overblik, beskrive og gruppere de mange mennesker eller organisationer, der har en interesse i det givne projekt.
Som det implicit også gives udtryk for, spænder interessenter så vidt, hvorfor det nødvendiggøre at organisere disse forskellige interessenter i grupper. Dette kan eksempelvis være i forhold til, i hvor høj grad interessenten[1]:

- har indflydelse på gennemførelsen af projektet
- bliver påvirket af projektet

Vi har derfor fire overordnede typer af interessenter:

- Eksterne interessenter – Lille indflydelse og ikke påvirket
- Gidsler – Lille indflydelse og påvirket
- Grå eminence – Stor indflydelse og ikke påvirket
- Ressourceperson – Stor indflydelse og bliver påvirket

Efter at have gennemtænkt projektets forløb, fra det spæde start til det er afsluttet, er vi kommet frem til flg, personer der kommer i berøring med dette.

- Projektejeren/erne
- Projektets initiativtager/e
- Eksterne interessenter, så som slutbrugere.

Projektejeren/erne - Ressourcepersoner
Ejerens projekt er vores kunde, der ønsker at vi udvikler et Monopoly Junior spil. Da de skal acceptere det endelige resultat af projektet, involveres de så meget og så ofte det kan lade sig gøre. Dette er med til at lette forståelsen af projektets overordnede formål og hjælper vores udviklere til at programmere

[1] https://altomledelse.dk/interessentanalyse/

et spil helt efter kundens krav og ønsker. Kunden har derfor både en indflydelse på projektet og bliver i den grad påvirket af projektet.
Det kan også tænkes at firmaets ejere og ledere, herunder projektlederen kan have delvis ejerskab over projektet. De står nemlig for et stort ansvar for færdiggørelsen af det givne projekt før det videreoverdrages til kunden, som herefter får det fulde ejerskab. Disse ledere agerer også som ressourcepersoner - der sørger for at stille nødvendige ressourcer til projektets tilblivelse - hvorfor disse involveres under hele projektets forløb. De har derfor også et stort indflydelse og bliver også påvirket af projektet. Kunden samt lederne i firmaet udgør derfor de vigtigste interessenter for projektet.

Projektets initiativtager/e - Ressourcepersoner
Initiativtagerne er virksomhedens ejer/e, herunder også projektlederen, der styrer og leder projektets team, som består af programmører og softwareudviklere og andre organisatoriske instanser (såsom økonomiafdelingen), der tager del og har et vis ansvar til implementering af projektet.
Projektlederen har styringen og har derfor en stor indflydelse på projektet og bliver derfor i den grad også påvirket af projektet.

Eksterne interessenter - Gidsler
I dette tilfælde kan det ønskes at kundernes kunder er eksterne interessenter, som det nærværende projekt skaber ny virkelighed for i form af Matador junior spillet. De får mulighed for at spille spillet i samvær med deres omgangskreds i hyggelige rammer. Da de bliver påvirket af slutresultatet, skal de derfor informeres undervejs, hvorfor som projektleder skal man bruge lidt energi samt ressourcer på disse.
De offentlige instanser såsom skat, bank og staten er nogle af de eksterne interessenter, der hverken har indflydelse på projektet eller bliver påvirket af projektet. Disse skal blot orienteres og vide at projektet eksisterer.

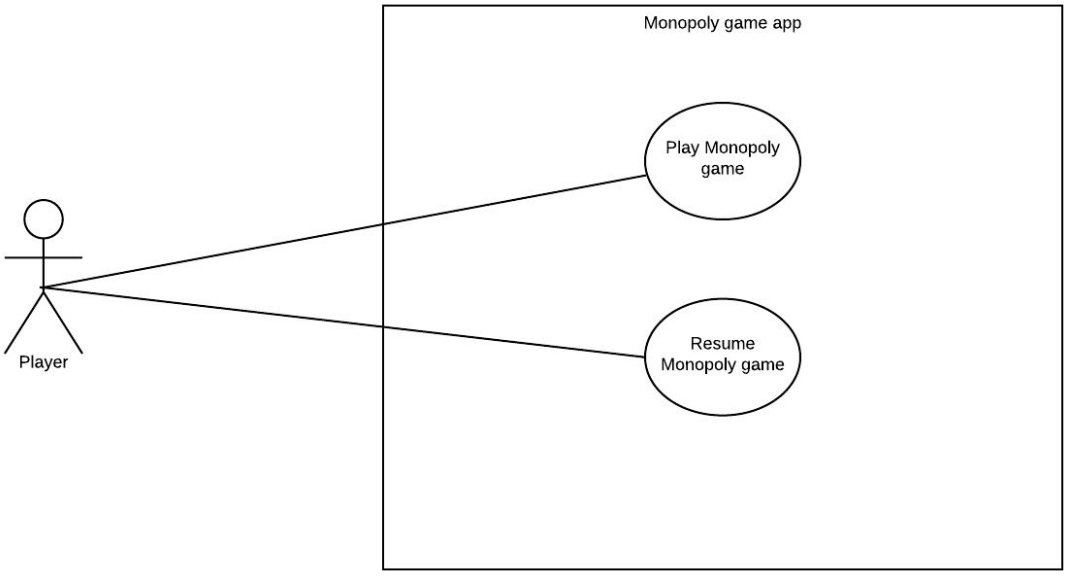
Udviklere samt øvrige medarbejdere - Grå eminence
Denne gruppe er ikke særligt interesserede i projektet, men de har indflydelse på det. Projektlederen skal derfor tage brug af ledelsesmæssige værktøjer der fremme motivationen blandt disse og sørge for at de alle sidder med følelsen af, at projektlederen tager dem alvorligt.
Interessenterne er nu blevet organiseret samt grupperet, som illustreres i tabellen nedenunder.

	Stor indflydelse	Lille indflydelse
Bliver påvirket af projektet	Virksomhedens ejer(e), Kunder, Projektleder(e)	Eksterne kunder(slutbruger)
Bliver ikke påvirket af projektet	Udviklere, øvrige medarbejdere	De offentlige instanser, Bank, Skat, Regeringen

Tabel 2: Grupperingen af projektets interessenter.

Use Case Diagram

Man bruger use case diagrammer til at vise hvordan en bruger kan interagere med det pågældende system. I udviklingen af Monopoly spillet har vi kun en use case, der består af



Billede 2: UML-diagram af use case

Use case UC1: Play Monopoly game

Hoved Actor: Player

Stakeholder and interests:.

- Player: interesserne for den player, der spiller Monopoly spillet, er at observere spillet i et format, hvor det er let at se og overskue, handlinger i monopoly gamet.

Precondition:
Mere end en spiller, minimumskrav er to spillere for spillet at spille. en person kan fungere som en eller flere spillere, hvis han vælger. maximum er fire spiller i spillet.

postcondition:
Vinderen annonceres, og monopolspil lukkes.

Main Success Scenario (Basic Flow):

1. Player anmodning initialisering af Monopoly spillet og indtaster antallet af spillere og spillernavne.
2. Player / spil begynder at spille.
3. systemet viser spil sporet, dvs. terningens værdi, spillerposition, felt for næste spiller træk, land på felt handling, spiller konto.
4. gentag trin 3, indtil en spiller ikke har nogen penge, og vinderen afgøres ud fra, hvem der har flest penge, eller spilleren annullerer / gemmer Monopoly spillet.

Alternativ flow:

1. Player anmodning initialisering af Monopoly spillet og indtaster et antallet af spillere som enten er for lidt eller for mange til at Monopoly spillet kan startes.
2. System stopper da og lukkes da Playerens handling har overskrevet min/max antallet af spiller og bedes prøve igen.

Use case UC2: Resume Monopoly game

Hoved Actor: Player

Stakeholder and interests:.

- Player: en interesse for den player, der spiller Monopoly spillet, er at kunne starte det gemte spil på et givet tidspunkt.

Precondition:
Et tidligere spil er gemt i en save file som kan resumeres af Monopoly gamet.

postcondition:
Spillet er resumeret og man kan fortsætte fra det gemte sted.

Main Success Scenario (Basic Flow):

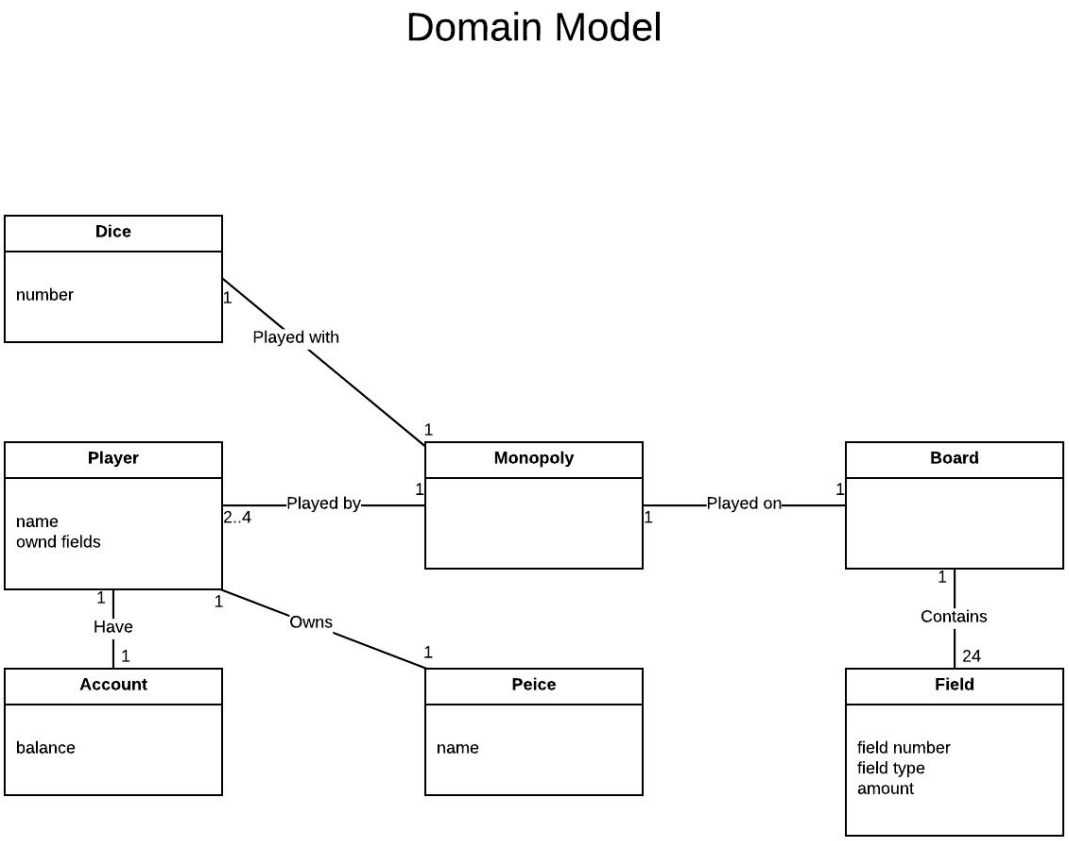
1. Player anmodning initialisering af Monopoly spillet og vælger resume game funktionen.
2. Player / spil begynder at spille videre fra det gemte sted
3. systemet viser spil sporet, dvs. terningens værdi, spillerposition, felt for næste spiller træk, land på felt handling, spiller konto.
4. gentag trin 3, indtil en spiller ikke har nogen penge, og vinderen afgøres ud fra, hvem der har flest penge, eller spilleren annullerer / gemmer Monopoly spillet.

Alternativ flow:

1. Player vælger resume game funktionen uden at der er et gemt spil(save file).
2. En system meddeles fortæller Player at spillet ikke kan resumeres på nuværende tidspunkt da der ikke er noget gemt spil(save file)

Domænemodel

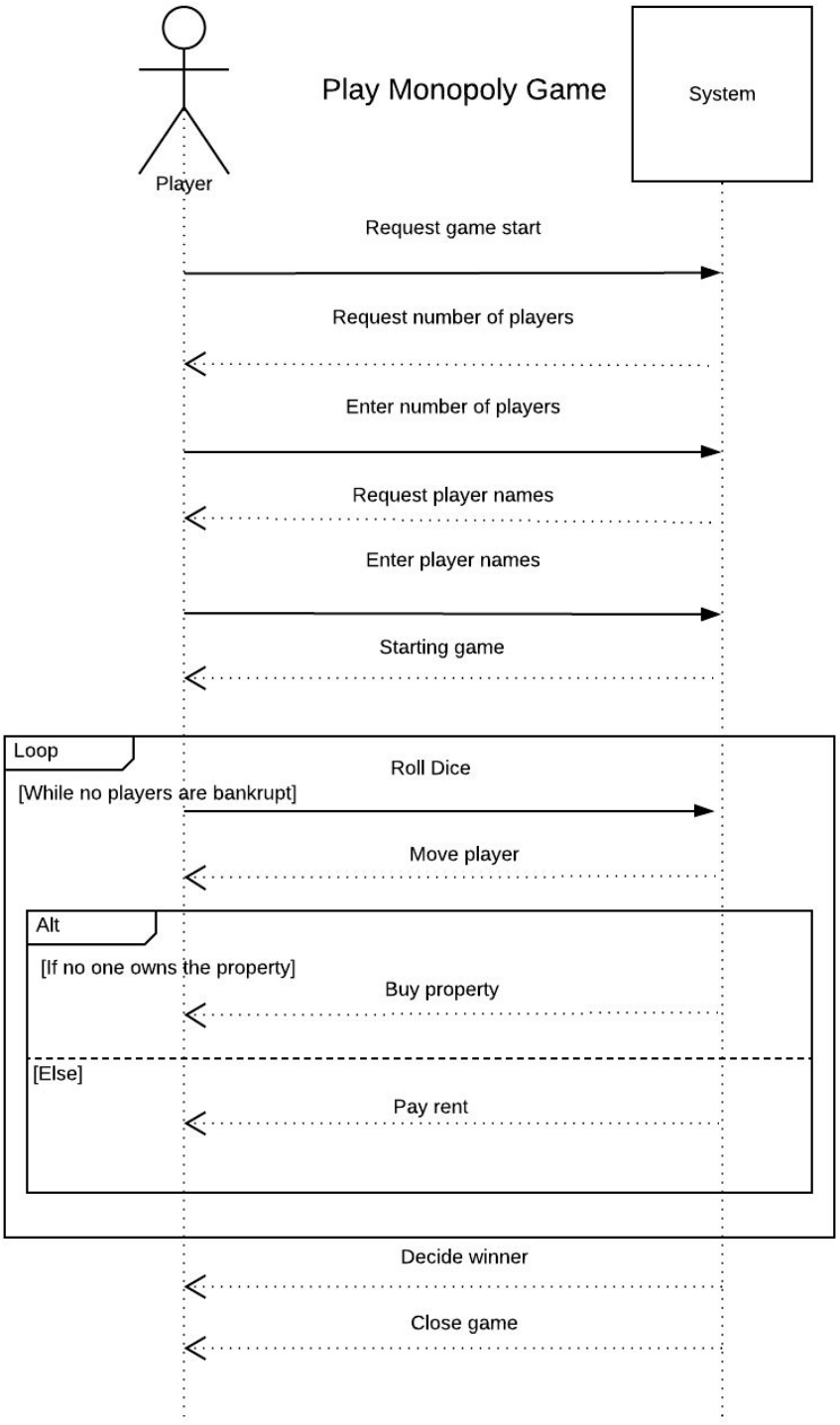
I det følgende har vi udarbejdet en domænemodel af spillets koncept samt formål. Dette blev gjort i lucidchart.
De valgte attributter afspejler kravene der er beskrevet ovenover. Eksempelvis havde vi kravene:
Dice (Terning): Spiller vil bevæge sig mod uret, ud fra terningkastets værdi. Terningens værdi efter kastet bestemmer briketts distance samt dets placering i et felt.
Field, amount: printer feltets værdi.



Billede 3: UML-diagram af Domænemodellen

En domænemodel er et visuelt model af domænet (i dette tilfælde vores spil) hvor man anvender både opførsel og data fra programmet til at vise domænet og hvordan det kommer til at fungere.

Systemsekvensdiagram



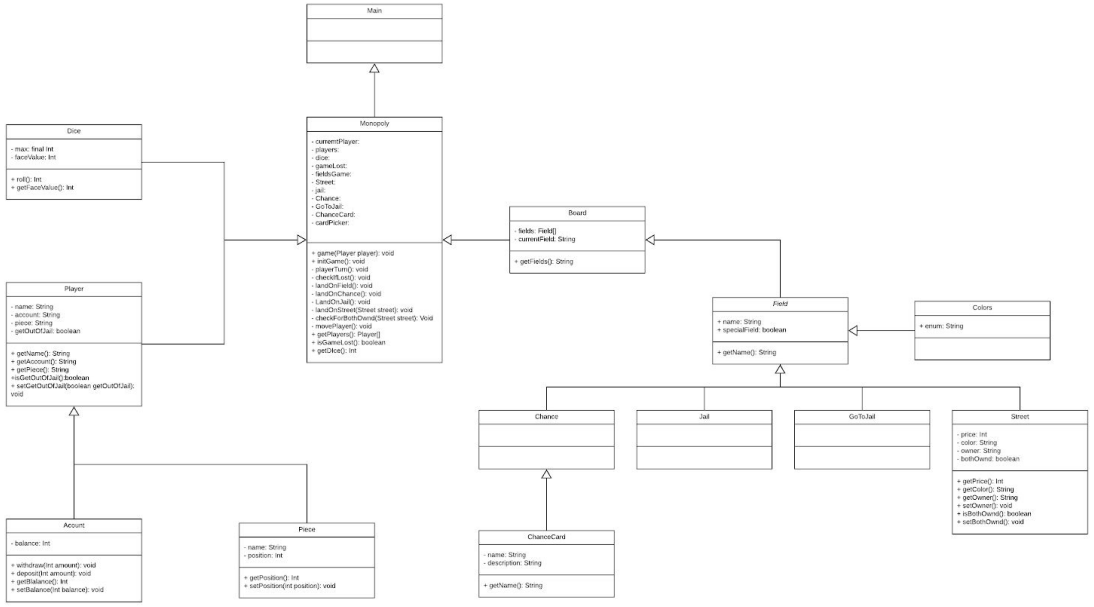
Billede 4: Systemsekvensdiagram(SSD). UML-diagram.

Et system sekvens diagram viser et specifik scenarie ud fra use casen. I vores tilfælde har vi valgt at gør det ud fra vores egen usecase som hedder play monopoly.

Design

Under objektorienteret design (eller ganske enkelt objektdesign) lægges der vægt på at definere software objekter. Det betyder at software-systemet betragtes indefra, hvor der tages udgangspunkt i mulige tekniske konstruktioner og fastlægger hvordan kravene realiseres på den tekniske platform[2]. Hertil anvendes der af softwaremæssige udviklingsmetoder og værktøjer i form af UML diagrammer samt beskrivelser.
I det følgende anvender vi statiske og dynamiske UML modeller til visualisering af spillets system.

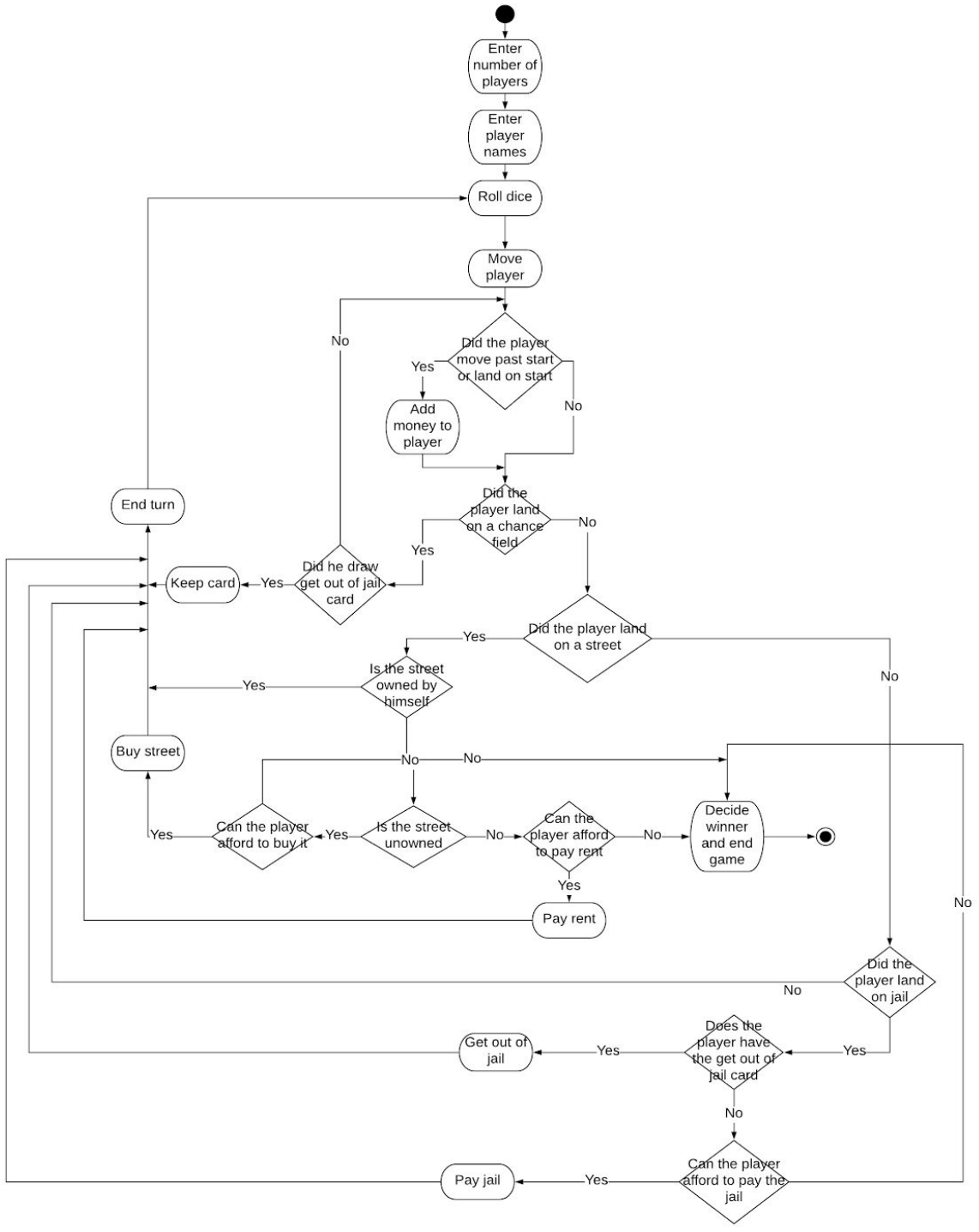
Design-klassediagram



Billede 5: Designklassediagram

[2] https://cn.inside.dtu.dk/cnnet/filesharing/download/8cb41957-4c65-4a78-acda-7e7d534e67be

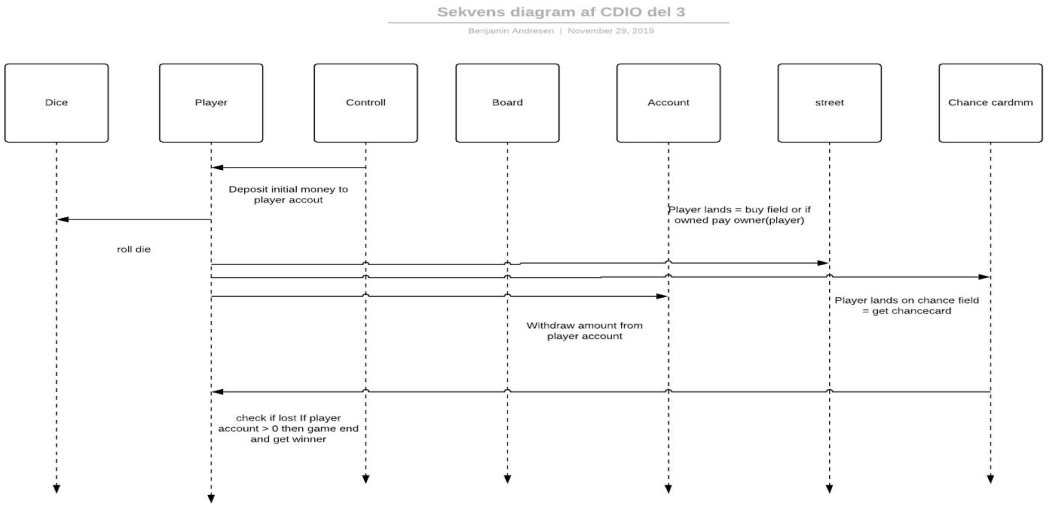
Aktivitetsdiagram



Billede 6: Aktivitetsdiagram

Et aktivitets diagram er en visuelt repræsentation af arbejdsflowet af aktiviteter i et system. det viser også flowet af data mellem aktiviteterne.

Sekvensdiagram



Billede 7: Sekvensdiagram(SD)

Det ovenstående diagram er lavet ved brug af hjemmesiden www.lucidchart.com. Diagrammet viser hvilket handlinger programmet foretager sig når Spilleren foretager sig noget.

Dokumentation

- Forklar hvad arv er.

Arv (Eng. :inheritance), er en af de vigtigste elementer indenfor OOP (Object-orienterede programmering).
Programmeringssproget Java, som er et OOP-sprog, benytter konceptet 'klasser', en slags skabelon, som angiver rammerne for et objekt.
Arv i Java, muliggøre at oprette underklasser, til en hovedklasse, der indeholder der samme attributter og metoder som hovedklassen.
Hovedklassen kaldes for *Super Class*, og indeholder altså de metoder og attributter som kan nedarves til andre klasser og objekter.
Underklassen, som nedarver fra Super Class, kaldes for *Sub Class*, og kan også tilføje sine egne metoder og attributter.

Konceptet bag arv i OOP, muliggøre genbrugelighed af allerede eksisterende kode, hvilket mindsker brug af hukommelse, og skaber et mere systematisk kodemæssigt overblik.

Nedarvningen kan vises således:

```
Class Sub-Class extends Super-Class
{
   //metoder og attributter
}
```

En Super Class kan have flere underklasser, men en Sub Class kan kun have en hovedklasse.
Sub Class arver ikke konstruktører fra Super Class'en, men konstruktør fra Super Class'en kan stadig blive kaldt fra Sub-Class'en.

- Forklar hvad abstract betyder.

En abstract klasse er en konceptuelt klasse, der indeholder eventuelle brugbar fællestræk, som kan benyttes af underklasser.
Forskellen mellem en normal og en abstract klasse, er at en abstract bruges kun som skabelon, og derfor kan en abstract klasse ikke instantieres eller laves som en objekt, hvorimod dette er tilfældet for en almindelig klasse.

Eksempelvis kan en abstract klasse, stå for de nødvendige basale rammer for udformningen af en cykel. Abstract-klassen vil derfor indeholde stel, hjul, bremser, lygter osv., men de yderligere specifikationer vil først blive specificeres i underklassen, såsom for Mountain-bike og City-Bike,
hvor blandt andet hjulstørrelsen af forskellige for de to cykler.

Abstracte klasser vises således:

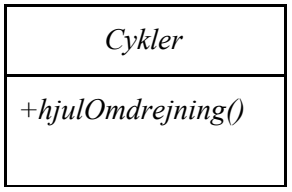
```
public abstract class Cykler
{

 //Abstrakte metoder & attributter kan skrives som abstract. En abstract klasse kan, men
 //behøver ikke at indeholde abstrakte metoder.
 //Abstrakte metoder kan kun fremkomme i abstracte klasser, og ikke i almindelige klasser.

        Abstract void hjulOmdrejning ();

}
```

Abstract klasser skrives med skråskrift/kursivt, når det fremvises grafisk på en diagram.
Sub-klasserne kan implementere metoden forskelligt, for hvert af de klasser.



- Fortæl hvad det hedder hvis alle fieldklasserne har en lanOnField metode der gør noget forskelligt.

landOnField er en abstract metode, hvis eneste formål er at skabe et fundament for andre felt/fieldklasser.
Vi har oprettet en abstract 'Field' klasse, som indeholder en abstract metode, der skaber et skelet, som kan implementeres i sub-klasserne, som repræsenterer de respektive felter på Monopoly-Junior brættet.
Dermed kan hvert felt, som er skabt ud fra den abstrakte Field klasse, indeholde den samme landOnField-metode, som hvert underklasse kan implementere og modificere til deres egne specifikationer.

GRASP

GRASP (General Responsibility Assignment Software Patterns/Principles), er et konceptuelt fremgangsmåde, til uddelegering af af ansvar og information til de forskellige klasser og objekter. Dette er en af grundelementerne for OOP (Objekt Orienteret Programmering)

Nedenfor, ses vores forsøg på anvendelse af GRASP:

Creator, er en klasse man bruger til at oprette nye instanser af objekter med, fx kan man lave en creator klasse for en bil, indeholder: årgang, model, foran(vh, hj) bagved(vh, hj) og således. vi bruger det f.eks i opgave til streets hvor vi laver en creator med navn pris og color som er en string, int og string.

Controller, er det komponent i systemet, der står for den logiske fremgangsmåde. Oftest kan det visuelt demonstreres, ved at gøre brug af controller klasse, der er under GUI (Graphical User Interface). Det er således Controllers opgave, at sørge for at bruger input bliver igangsat korrekt. Monopoly-klassen er den klasse der står for meget af den logik, for hvordan spillet skal køres, hvorimod main, står for initiering af systemet.

Information Expert princippet, har forsøgt at blive efterlevet, ved at tildele de fornødne ansvarsområder, med udgangspunkt fra hvad de respektive klasser har af information.

Polymorphism, har vi gjort brug af i Field-klassen, som nedarver dens komponenter til Chance-klassen, Jail-klassen, goToJail-klassen, og ikke mindst, Street-klassen.

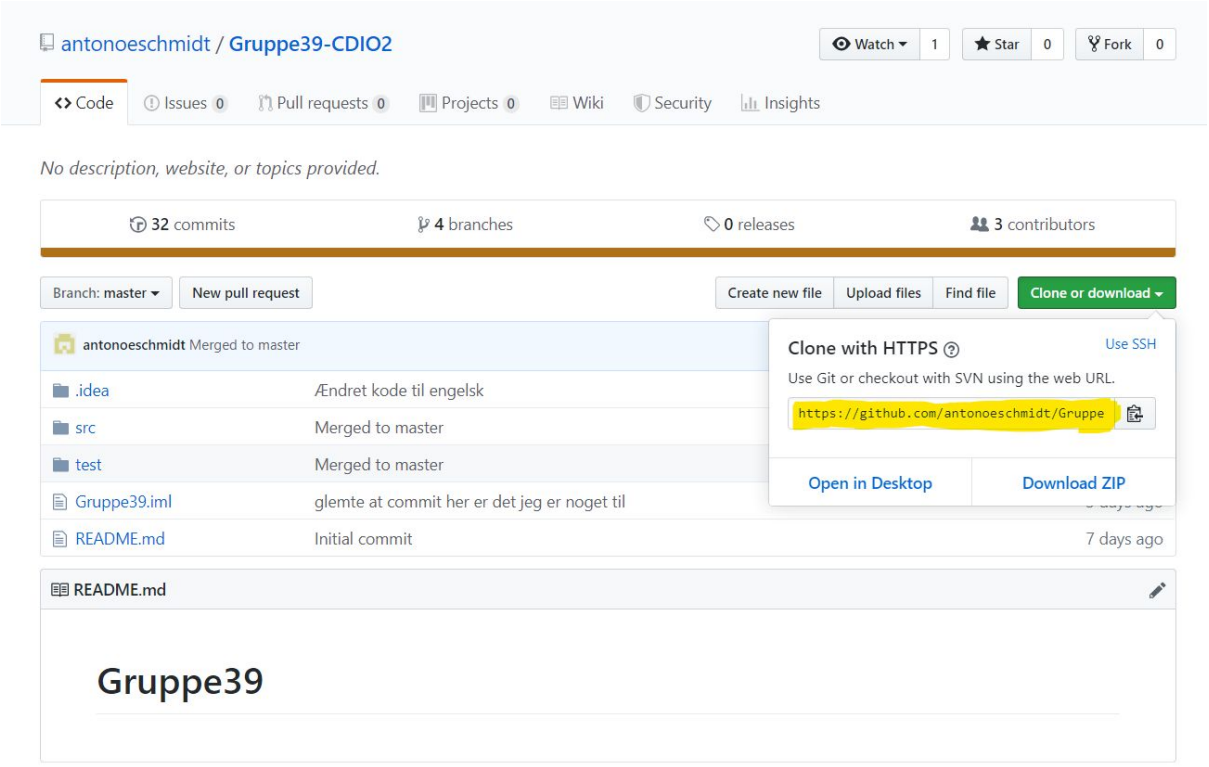
Lav kobling, har vi forsøgt på at implementer, i blandt andet i Account-klassen, således at ændring i de associeret klasser, vil have mindst påvirkning på systemet, som muligt.
Dice-klassen er endnu et eksempel på dette, da ændring i dens kode, vil have relativt lille påvirkning, da klassen står meget isoleret.
Lav kobling er ønsket, men altid muligt, og Monopoly-klassen er en del højere kobling, grundet dens associeringer med de andre klasser, samt fordi den har den primærer ansvar for at styre spillet.

Høj binding, er blevet implementeret ved fordeling af ansvarsområder ud på tilstrækkelig mange klasser, samt holde metoder så simple som muligt. Blandt andet har klassen Player, fordelt ansvaret for funktionaliteten, ud på Account og Piece klassen, hvilket skaber en mere sammenfattende sammenhæng for videreudviklingen af koden.

Versionsstyring

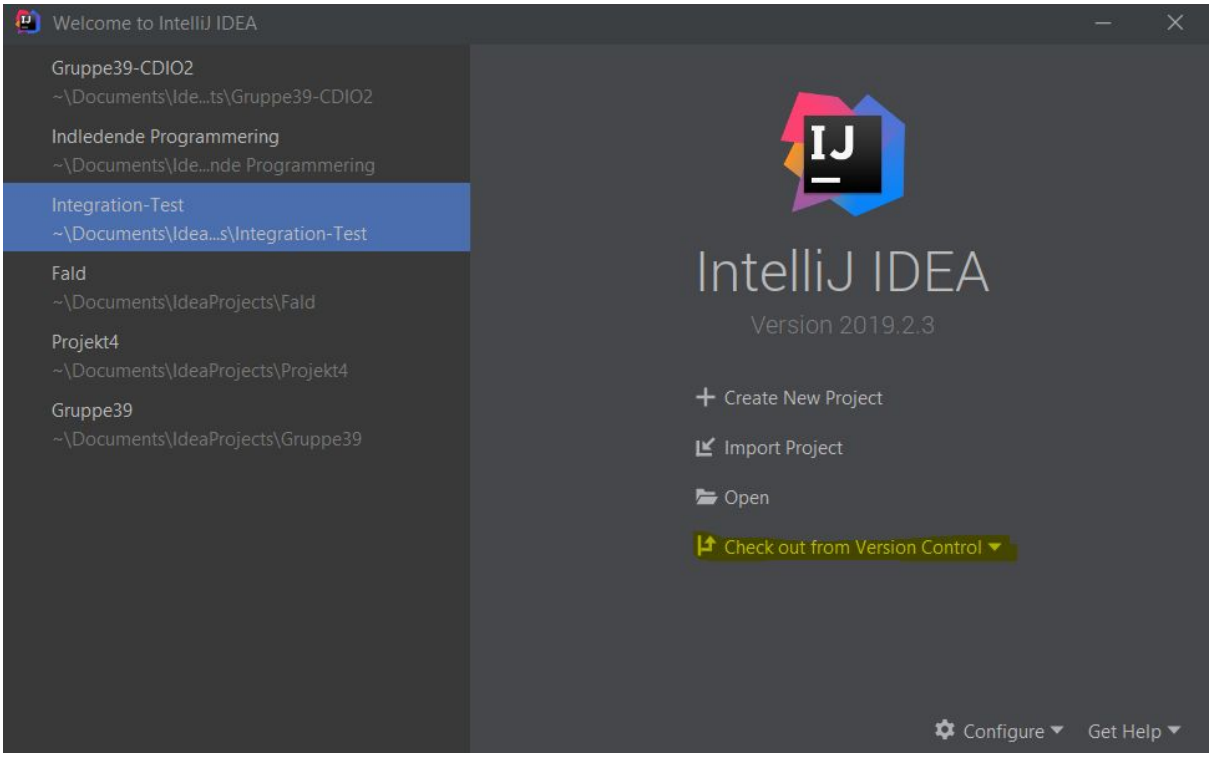
Github:
For at få adgang til Github repository, gør følgende:

1.
Kopier linket der er markeret med gult.



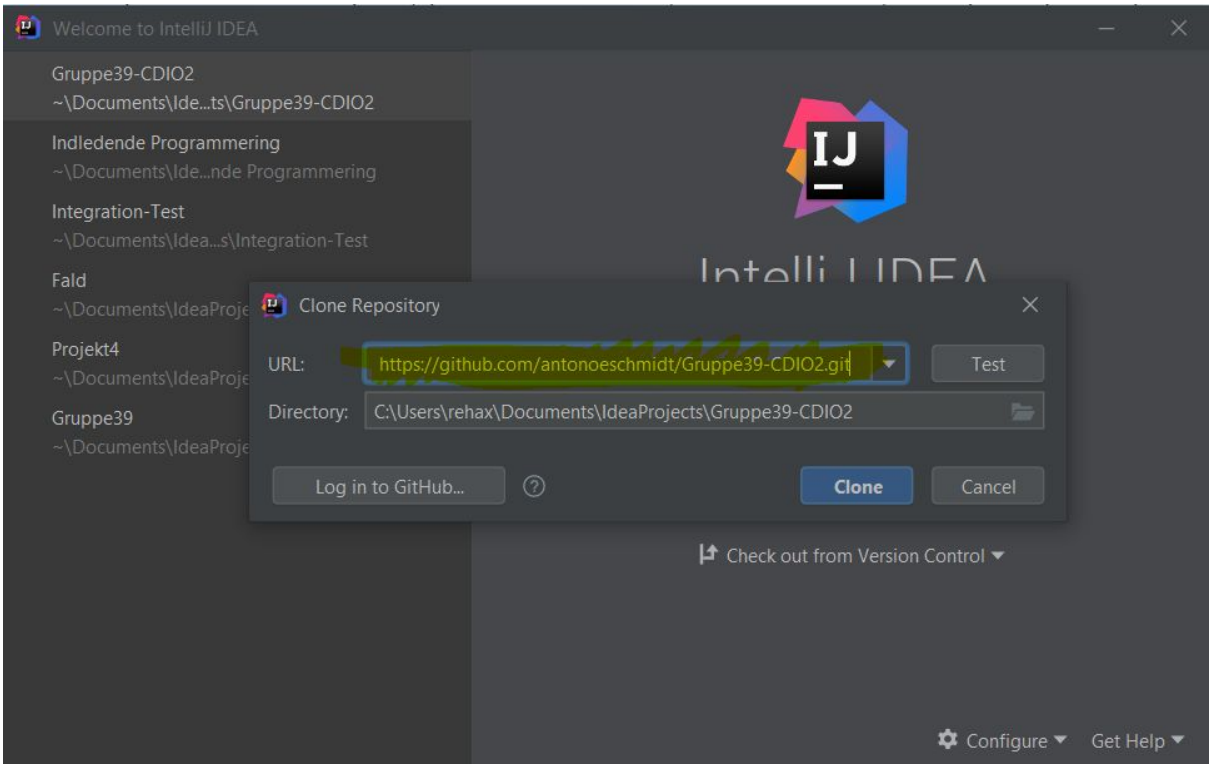
Billede 8: Linket til Git-repository.

2.
Åben IntelliJ og åbn menuen *"Check out from Version Control*", og vælg derefter "*Git*"



Billede 9 - IntelliJ startside

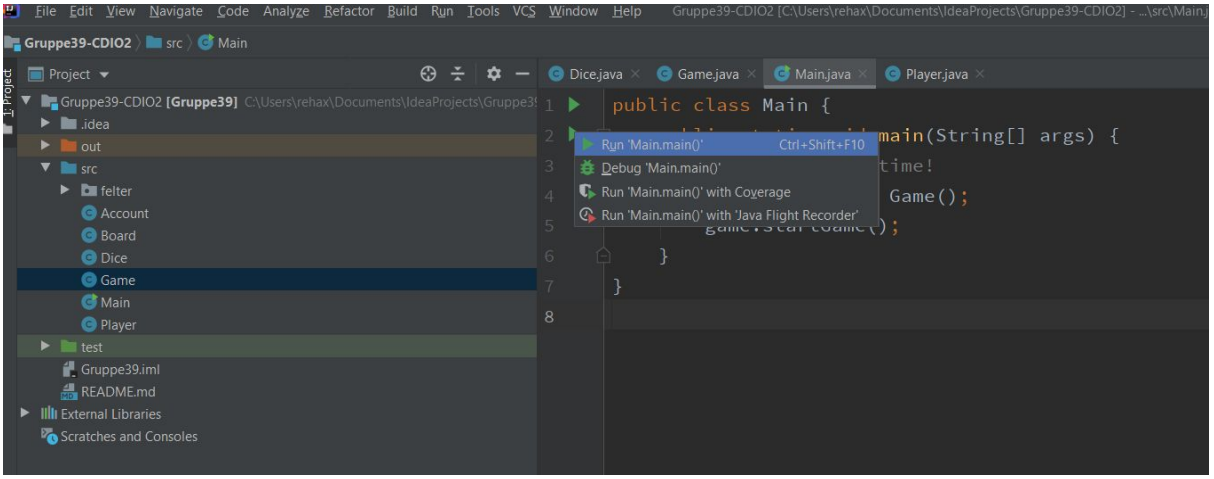
3.
Og indsæt nu det kopieret link fra Github repo'en ind i URL sektionen i IntelliJ.



Billede 10: illustration af opsættelsen af GitHub repo.

IntelliJ:
For at bruge IntelliJ til at starte spillet, gør følgende:

1.
Under "src"- stien, vælg "Game" klassen og tryk på den grønne pil, og vælg "Run 'Main.main()' "



Billede 11 - Main metoden.

Konfigurationsstyring

Denne sektion gennemgås de hardwaremæssige, samt de softwaremæssige krav til afvikling af programmet. Ydermere vil kunden finde en grafisk vejledning til opsætning af visse påkrævet programmer.

Minimumskrav til hardware, for afvikling af JAVA koden. :

(Kilde:https://www.java.com/en/download/help/sysreq.xml)

WindowsWindows 10 (8u51 and above)

- Windows 8.x (Desktop)
- Windows 7 SP1
- Windows Vista SP2
- Windows Server 2008 R2 SP1 (64-bit)
- Windows Server 2012 and 2012 R2 (64-bit)
- RAM: 128 MB
- Disk space: 124 MB for JRE; 2 MB for Java Update
- Processor: Minimum Pentium 2 266 MHz processor
- Browsers: Internet Explorer 9 and above, Firefox

Mac OS X

- Intel-based Mac running Mac OS X 10.8.3+, 10.9+

- Administrator privileges for installation
- 64-bit browser

A 64-bit browser (Safari, for example) is required to ru
n Oracle Java on Mac.

Linux

- Oracle Linux 5.5+[1]
- Oracle Linux 6.x (32-bit), 6.x (64-bit)[2]
- Oracle Linux 7.x (64-bit)[2] (8u20 and above)
- Red Hat Enterprise Linux 5.5+[1], 6.x (32-bit), 6.x (64-bit)[2]
- Red Hat Enterprise Linux 7.x (64-bit)[2] (8u20 and above)
- Suse Linux Enterprise Server 10 SP2+, 11.x
- Suse Linux Enterprise Server 12.x (64-bit)[2] (8u31 and above)
- Ubuntu Linux 12.04 LTS, 13.x
- Ubuntu Linux 14.x (8u25 and above)
- Ubuntu Linux 15.04 (8u45 and above)
- Ubuntu Linux 15.10 (8u65 and above)
- Browsers: Firefox

For afvikling af programmet, kræves følgende software installeret; ***Java SE Development kit 1.8*** og ***JetBeans IntelliJ*** anbefales som IDE til at køre koden som en text-based-Interface.
Udover dette kræves **internetadgang**, for at kunne downloade programmet fra **Github repository**
(Link til ovennævnte kan fremfindes ved at klikke på nedenstående blå elementer)

Installation af IntelliJ:

En guide til installation af JetBeans IntelliJ kan findes her:
https://www.jetbrains.com/help/idea/installation-guide.html
(OBS: minimumskravene for IntelliJ er højere end kravet for kravene for Java)

Installation af SDK:

https://codenotfound.com/java-download-install-jdk-8-windows.html

Test

Vi har lavet 3 forskellige test.

Testcase 1

Test case ID	TC1
Dato	
Testet af	
Testmiljø	f.eks Macbook Pro 2015, Catalina 10.51.1
Beskrivelse	Vi skal være sikker på at vores terning math random er random ved hjælp af en test kan det testes hvis terningerne slås 6000000 slag skal ca 1000000 slag laden på værd af de mulige øjne som terningen viser fra 1-6 men en afvigelse på 5% acceptabel.
Test krav	Der må ikke forkomme en størrer afviggelse ind 5%. det vil sige at hvis der bliver slået 1 mindre/mere end 950000/1050000 slag.
Før-betingelser	Vi har lavet en “dice” men roll funktionen med min=1 og max= 6 værdierne.
Efter-betingelser	
Test Procedure	f.eks: 1. Lav J-unit test 2. Opskriv testkode
Test data	De dataer/parameter vi bruger
Forventet resultat	er at resultatet passer overens med den teoretiske værdi
Aktuel resultat	Den passer overens med den teoretiske værdi
Status	bestået

Testcase 2

Den anden test har vi lavet var en brugertest. vi fandt X-antal forskellige mennesker med meget varierende alder for at tjekke om de kunne finde ud af programmet uden en manual. Det eneste krav er ingen af deltagerne må have kendskab til programmering

Test case ID	TC2
Dato	24-11/2019
Testet af	Benjamin s195828
Testmiljø	f.eks Macbook Pro 2015, Catalina 10.51.1
Beskrivelse	Denne test er for beregnet til at se om et almindeligt menneske kan finde ud af at spille spillet.
Test krav	ingen af deltagerne må have kendskab til programmering
Før-betingelser	Vi har lavet en Prototype af spillet som de kan prøve sig frem med.
Efter-betingelser	
Test Procedure	f.eks: 1. få mennesker til at prøve spillet 2. observere test deltagerne
Test data	De dataer/parameter vi bruger
Forventet resultat	ca. 80 kan finde ud af spillet
Aktuel resultat	alle kan finde ud at spille spillet
Status	bestået

kommentar	Alder	Køn	Kunne personen finde ud af det?	Uddannelse
spillet var kedeligt	47	Kvinde	Ja	Folkeskole
	21	Mand	Ja	gymnasium
hvad er pointen med spillet?	67	Kvinde	Ja	gymnasium
	22	Mand	Ja	10.klasse
	24	Mand	Ja	Folkeskole
	22	Mand	Ja	gymnasium
	59	Mand	Ja	Universitet
	29	Mand	Ja	Folkeskole (special)
	32	Kvinde	Ja	2 semester universitet
det er ikke sjovt det her	11	Kvinde	Ja	Folkeskole

Tabel 3: Skema over test deltager

Testcase 3

Test case ID	TC3
Dato	29-11/2019
Testet af	Anton s163053
Testmiljø	f.eks Macbook Pro 2015, Catalina 10.51.1
Beskrivelse	Denne test er beregnet til at tjekke om account klassen i spillet fungere rigtig. den tester at den ikke går i minus og at når man prøver at tilføje penge til kontoen at de faktisk bliver tilføjet.
Test krav	Balancen på account må ikke gå under 0, og skal kunne modtage depositum og withdraw metoderne.
Før-betingelser	Vi har lavet en account klasse som indeholder en balance som kan gå op og ned men ikke under 0.
Efter-betingelser	
Test Procedure	f.eks: 3. Lav J-unit test 4. Opskriv testkode
Test data	De dataer/parameter vi bruger
Forventet resultat	vores konto starter på 20 og vi withdraw 2000 og vi forventer der står 0 på kontoen efter. vores konto starter på 20 og vi despotier 100 og forventer der står 120 på kontoen efter.
Aktuel resultat	withdraw funktionen for den til at gå i 0 og ikke under. deposit funktionen for den til at gå til 120 og ikke mere.
Status	Bestået

Projektforløb

Projekters forløb har forgået godt. Der har været en lille smule problemer med vores koordinering og problemer med at overholde vores tidsplan. Derudover burde vi nok bruge mere tid på at poler vores program og vores rapport. Et eksempel på det er at vi ikke kunne nået at tilføje en load game og save game funktion. De ting kunne vi havde nået hvis vi havde planlagt vores arbejde bedre og var begyndt tidligere. Vi har lært fra tidligere projekter at vi skal bruge mere tid på vores arbejde og planlægge det bedre. I de tidligere projekter lærte vi på den hårde måde at vi skal bruge mere tid og i dette projekt har vi lært på den hårde måde at det nytter ikke kun at arbejde længere. Man skal også arbejde smarter.

Konklusion

For dette projekt har vi konkluder at vi kunne danne nogle krav for spillet baseret på kundens vision og ønsker. Det er med til at give et godt overblik over hvad der bliver forventet af vores program og hvor langt vi er kommet kodningen. Efter det visualiserede vi koden i form af UML diagrammer. Disse diagrammer og krav har hjulpet os med at skabe fundamentet for vores program. Det sidste vi kan konkludere at vi har lavet et program mellem 2-4 spiller som overholder alle vores og kundens krav, ønsker og visioner.

Bilag

Bilag 1 Timeregnskab

Dato	Navn	Emne	Antal timer
29/11-2019	Reza	Dokumentation Programmering	5 4
29/11-2019	Anthony	Dokumentation Programmering	4 5
29/11-2019	Sid	Dokumentation programmering	6 3
29-11/2019	Benjamin	Dokumentation programmering	3 7
29-11/2019	Anton	Dokumentation programmering	2 8
29-11/2019	Anders	Dokumentation programmering	5 4

Tabel 1 - Antal timer de enkelte medlemmer har bidraget med.