CDIO del 2

02312-14 Indledende Programmering

02313 Udviklingsmetoder til IT-Systemer

02315 Versionsstyring og Testmetoder

Gruppe nr: 44

Denne rapport indeholder 34 sider inkl. bilag og denne side.

Grp. 44 - Medlemmer:

Emil Lundqvist
(s185131)

Kristian N. Andersen
(s185125)

Armandas Rokas
(s185144)

Simon G. Philipsen
(s163595)

David Milutin
(s160601)

Abstrakt

The following report a project report made in addition to a program, developed for an assignment asking for a certain small version of a monopoly game with required documentation.

This project documentation will start with a small explanation of the assignment, followed by a mentioning of interested and involved parties. Thereafter, the assignment requirements will be listed and somewhat categorized for a better overview.
Following, is a conceptual analysis of the main scenario required to play out in the program written in the form of a use case. More of the concept is illustrated and explained, before changing the focus point from a real world concept to a more specialized, programming point of view, approach.
Later on in the project report, the actual implementation process and strategy is explained and partly illustrated. Afterwards, tests of the important parts of implemented game program is presented and explained.
Concluding the report an evaluation of the project is presented before a short conclusion winds up the project.

Indholdsfortegnelse

Indledning

I følgende rapport vil vi beskrive konstrueringen af et simplificeret 'monopoly spil', udviklet ud fra krav fra DTU, som har bestilt spillet for, at det kan implementeres på deres computere i Databar. Programmet bliver udviklet, og arbejdsprocessen beskrives i sammenhæng med præsentation, dokumentation og selve programmet, der udvikles gennem rapporten. Sidst vil det udviklede 'monopoly spil' afleveres sammen med rapporten, der fungerer som dokumentation til fremgangsmåden under udviklingen af programmet.

Analyse

Vision

Projektet går ud på at videreudvikle terningspillet, som blev lavet til IOOuterActive i et forudgående projekt. Spillet skal fortsat være mellem to personer, og med to klassiske terninger, men denne gang har spillerne en pengebeholdning. Spillerne kan forøge, formindske eller beholde de samme værdier i deres pengebeholdning efter hver kast. Det afhænger af, på hvilket af de 11 felter spilleren lander. Spillerne har et startbeløb på 1000, og vinderen er en spiller, som opnår 3000 eller derover først.

Interessent- og aktøranalyse

Interessent-mål tabel	
Interessent	**Mål**
IOOuterActive	At modtage et velfungerende spil med lave omkostninger og ikke mindst til tiden.
Spiller	At spille et underholdende spil.

Aktør-rolle tabel	
Aktør	**Rolle**
Spiller	Primær

Kravspecifikationer

K1: 2 spillere slår på skift.
K2: Spillet spilles med 2 terninger.
K3: Spillepladen har felter, som har værdierne 2-12, med et unikt scenarie til hvert felt (se feltoversigt).
K4: Spillerne starter på 1000 ‘penge’.
K5: Spillet slutter når en spiller når 3000 ‘penge’.
K6: En spillers balance må ikke blive negativ. Så balancen kan altså som det mindste være 0.

Supplerende specifikationer

Usability

K7: Der skal gives en liste af nødvendighedskrav til styresystem og installerede programmer, for at spillet kan køres som program.
K8: Vejledning i hvordan kildekoden compiles, installeres og afvikles. Inkl. beskrivelse af import fra git repository.

Supportability

K9: Spillet skal let kunne oversættes til andre sprog. Det betyder, at man skal kunne oversætte sproget uden brug IDE.

Performance

K10: Response time skal være mindre end 100 ms[1].

Hardware

K11: Spillet skal virke på maskinerne i DTU's databarer.

Software

K12: Java version 8.

Recoverability

K13: Man skal kunne følge med i hvordan spillet er blevet udviklet (GIT).
K14: Der skal committes hver gang en delopgave er løst eller forbedret, min. én gang i timen.

Development constraints

K15: Spillet skal overholde GRASP.
K16: Spillet skal være afprøvet og kunden skal kunne gentage afprøvningen (Forslag: JUnit tests). Skal indeholde test for: At balancen aldrig kan blive negativ.

[1] https://www.nngroup.com/articles/response-times-3-important-limits/

Use-case Beskrivelser

Casual use case

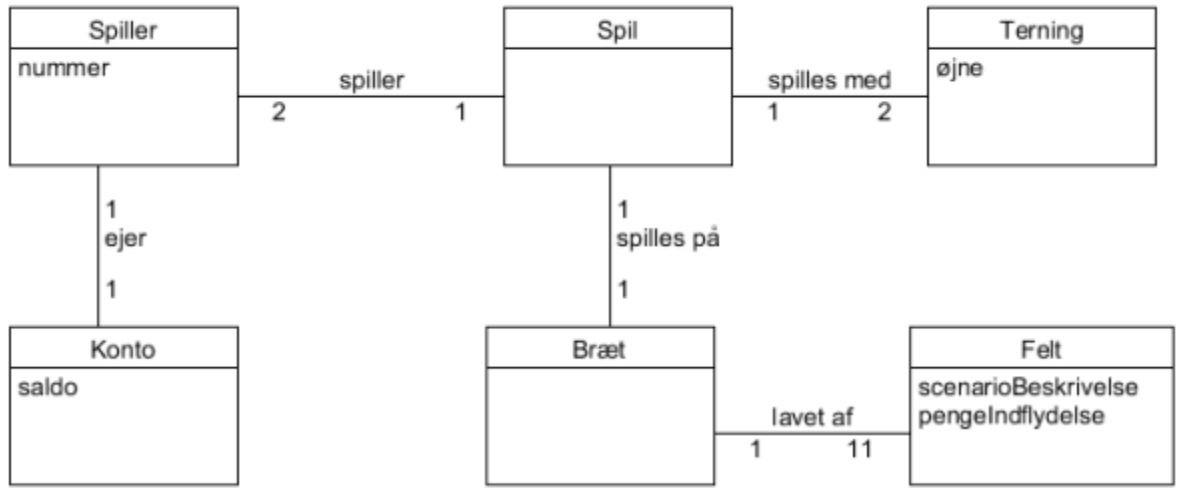
	UC1: Spil terningspil
Primær scenarie	Hver spiller starter spillet med 1000 penge. De slår terningerne skiftevis og lander på et felt, der bestemmer en ændring af deres pengebeholdning. Spillet slutter når en spiller har nået 3000 point eller mere.

Fully-dressed use case

	UC1: Spil Spillet
Scope	Terningespil
Level	User goal
Primær aktør	2 spillere
Stakeholders og interessenter	Spiller, IOOuterActive
Preconditions	Spillet er startet, og hver spiller har en pengebeholdning på 1000.
Postconditions	En af spillerne har nået en pengebeholdning på 3000 eller over. Alle pengeværdier bliver vist i en samlet stilling
Basic Flow	1. Spiller starter spillet. 2. Spiller kaster terningerne. 3. Systemet viser resultat for kastet og viser det felt spilleren er landet på, og giver en beskrivelse hvad der sker med spillerens pengebeholdning, Skifter til næste spiller. *Spillerne gentager punkt 2-3 indtil en vinder er fundet.* 4. Spillet viser sammenlagt resultat for begge spillere og hvem vinderen er.
Alternate Flow	3a Spilleren lander på felt 10. 1. Spilleren gentager punkt 2.
Teknologi og Data variations List	UTF-8 som tegnsæt Java 8 LTS: JRE
Kørselsfrekvens	1 gang pr. spil

Domænemodel

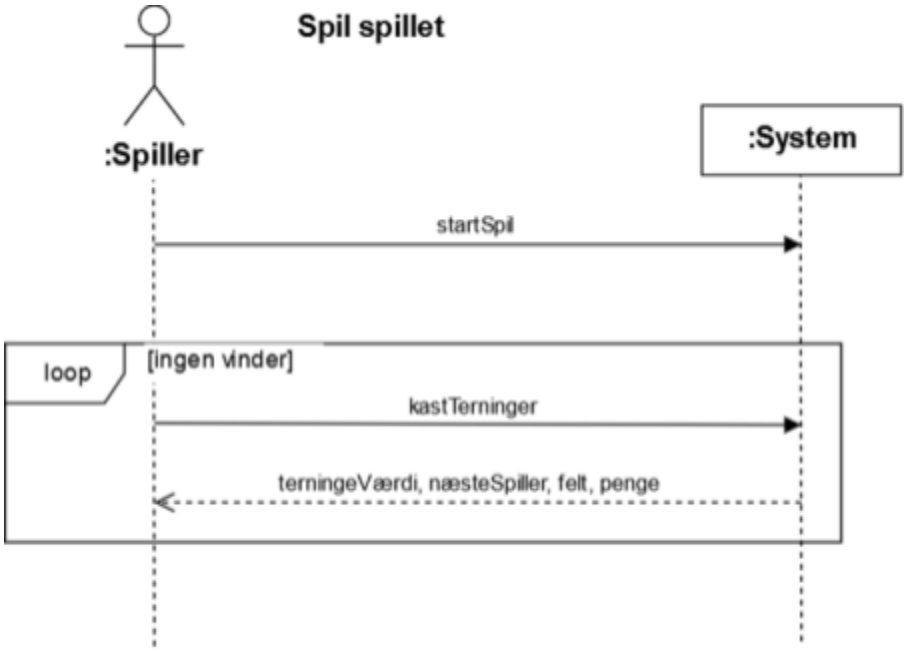
Diagrammet nedenfor beskriver koncepterne i spillets domæne. Det viser spillet, som spilles af to spillere, spiller 1 og spiller 2, som beskrives med spillerens nummer. De ejer også en konto hver, som indeholder information om deres pengebeholdning. Spillet spilles med to terninger, som viser et tilfældigt antal øjne hver gang man slår med dem. Desuden spilles spillet på et bræt, som er lavet af elleve felter, der hver har et skriftligt scenarie og en penge indflydelse på spilleren, hvis en spiller lander på dette felt.



Figur 1: Domain Model

Systemsekvensdiagram

Følgende er en illustration af hvordan 'spiller'-aktøren[2] arbejder med systemet. Dette skal læses oppefra og ned, hvor pilene følges, og forløbet indenfor 'loop'[3] gennemkøres indtil en spiller har vundet spillet.



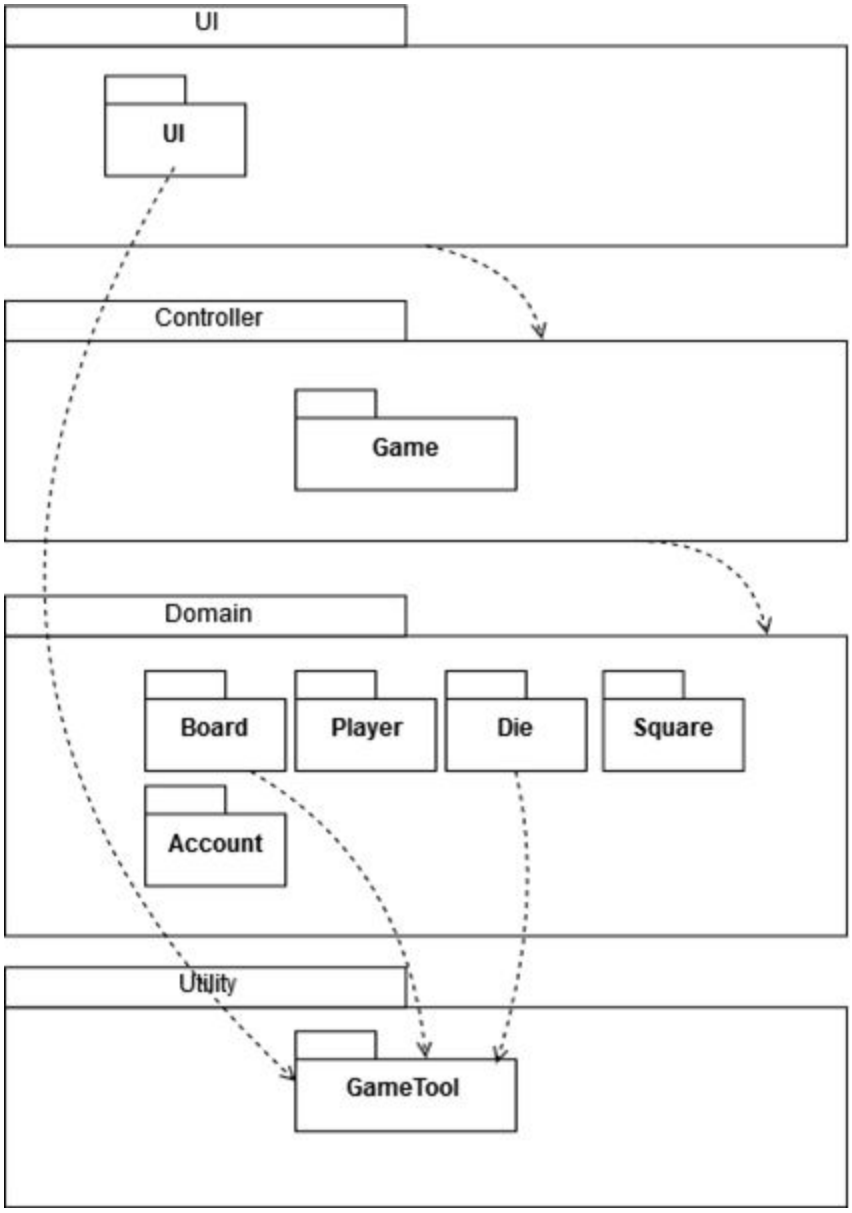
Figur 2: SSD over usecase "Spil spillet"

[2] Se ordbog - aktør
[3] Se ordbog - loop

Design

Package diagram

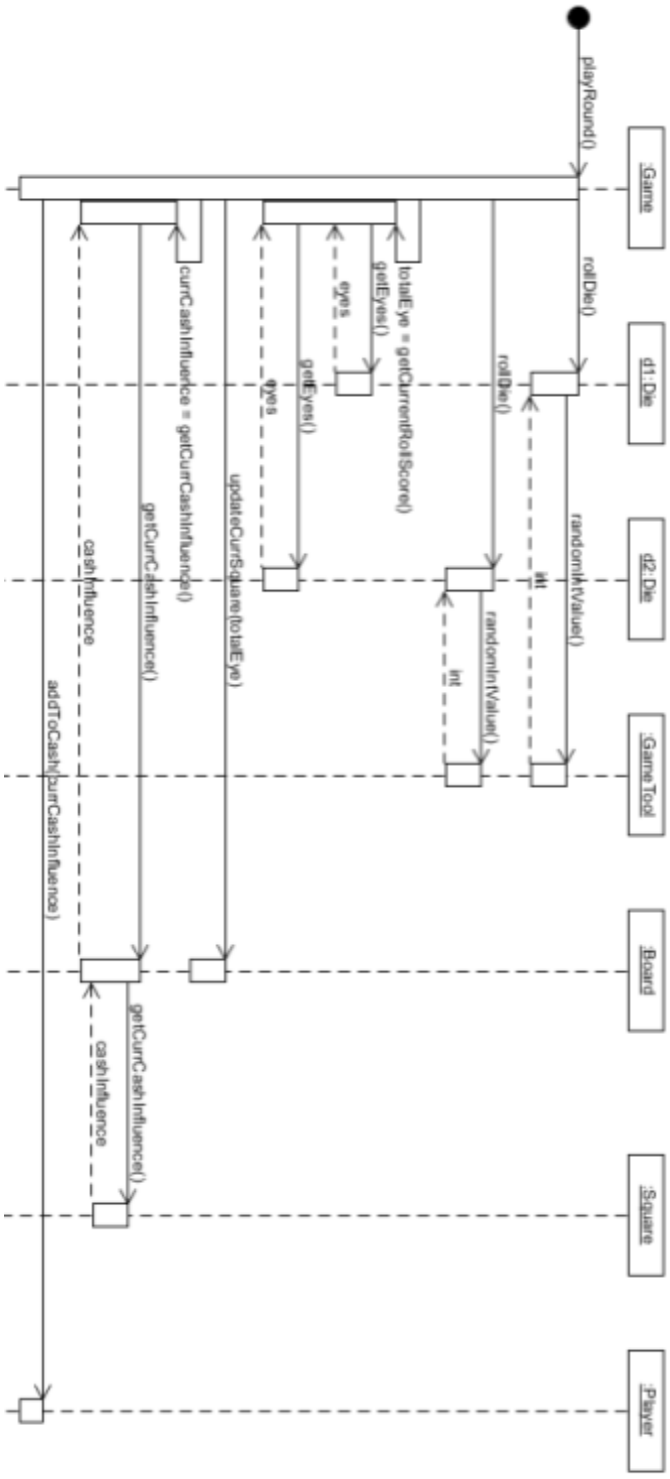
I pakkediagrammet skaffes der et overblik over hvilke pakker man regner med at bruge. Dette skal give forståelse af den generelle opbygning af programmet, og hvilke dele af indre pakker man vil tage i brug.



Sekvensdiagram

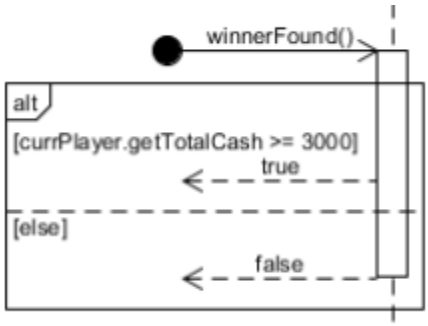
For use casen “Spil Spillet” kommer der kun til at være en enkelt interaktion mellem system og kunden. Dog er dette, for praktisk skyld, delt op i 3 dele: Spillerens runde, kontrol om han har vundet og, hvis han ikke har vundet, skift af spiller. Da dette bliver til et længere sekvensdiagram, er det blevet delt op i disse 3 dele i selve rapporten. Det fulde sekvensdiagram kan ses enten under bilag til rapporten (Bilag 1), eller findes som UML fil vedhæftet i ZIP filen med programmet.

Første del (Figur 4) omhandler spillerens tur. Her bliver der rullet med terningerne med hjælp fra vores egen pseudotilfældige metode i vores GameTool klasse. Der vil derefter skulle kommunikeres over det meste af programmet for at gennemgå turens effekt på spilleren. Der forventes ingen retur på dette stadie af spillet



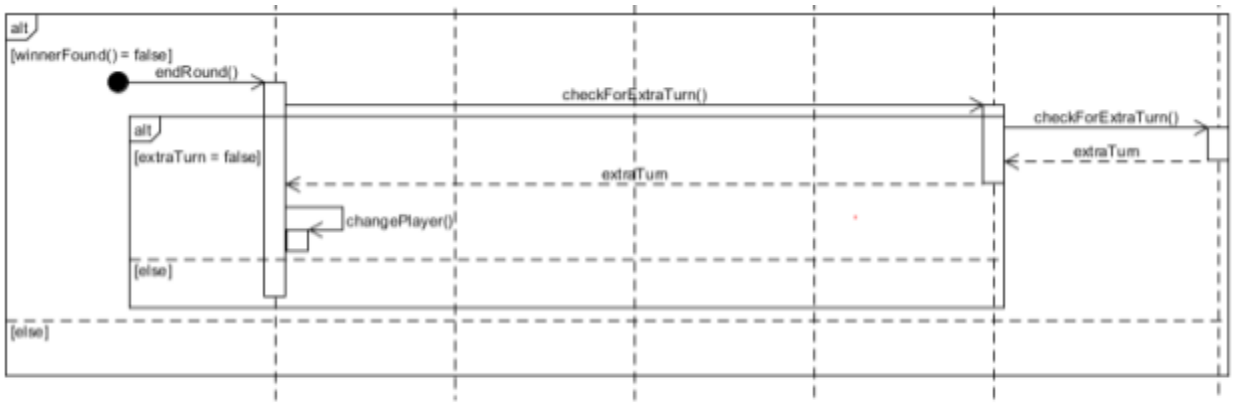
Figur 4: Sekvensdiagram del 1 ud af 3; playRound() metode

De næste kald fra UI til controlleren ‘Game’ sker uafhængig af input fra spilleren. Dette starter med en kontrol af vinderen (Figur 5), som via spillerens saldo ser efter om spilleren har vundet.



Figur 5: Sekvensdiagram del 2 af 3; winnerFound() metode

Til sidst kan vi se et kald fra UI om at sætte den næste spiller op, hvis der ikke allerede er blevet fundet en vinder (Figur 6). Hvis der skal skiftes tur, kontrolleres der først om spilleren er landet på felt 10, da dette udfald giver en ekstra tur. Altså skiftes der kun til den nye spiller, hvis den forrige spiller ikke er landet på felt 10.

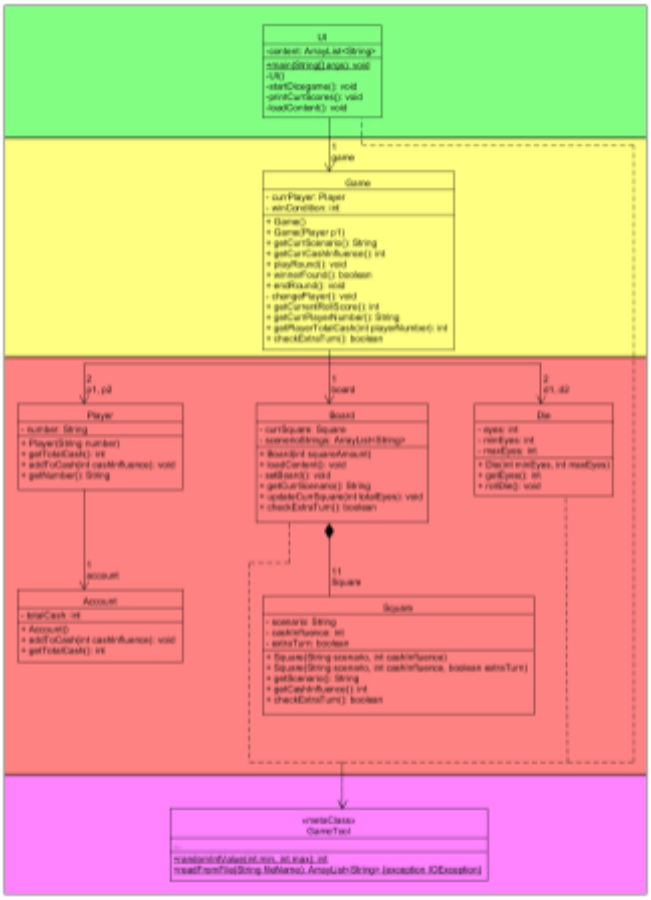


Figur 6: Sekvensdiagram del 3 af 3; endRound() metode

Design Klasse Diagram

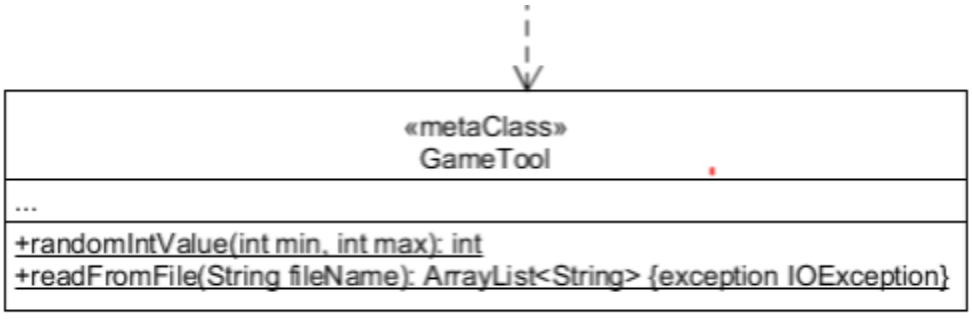
Design Klasse Diagrammet viser overblikket over hele programmet, hvilket også gør det til et af, hvis ikke det største diagram, der bliver lavet. Grundet dette vil der kun blive nævnt sammenhængen af lagene i systemet og et par af klasserne her, men vil man se det hele kan det findes under bilag i rapporten (Bilag 2), og yderligere findes som UML fil vedhæftet i ZIP filen med programmet.

Til højre (Figur 8) kan der ses hvordan lagene er delt op. Her er det grønne lag, hvor UI ligger, og dette er laget som aktøren har adgang til. I gult ligger vores controllers, som står for kommunikationen mellem UI laget og informationen, der ligger i domænelaget. Domænelaget er laget lige under og er vist med rød farve. Her ligger klasserne som systemet skal bruge, for at instantiere objekter det kan interagerer med. Til sidst er det fjerde lag, som er 'technical services' for programmet.



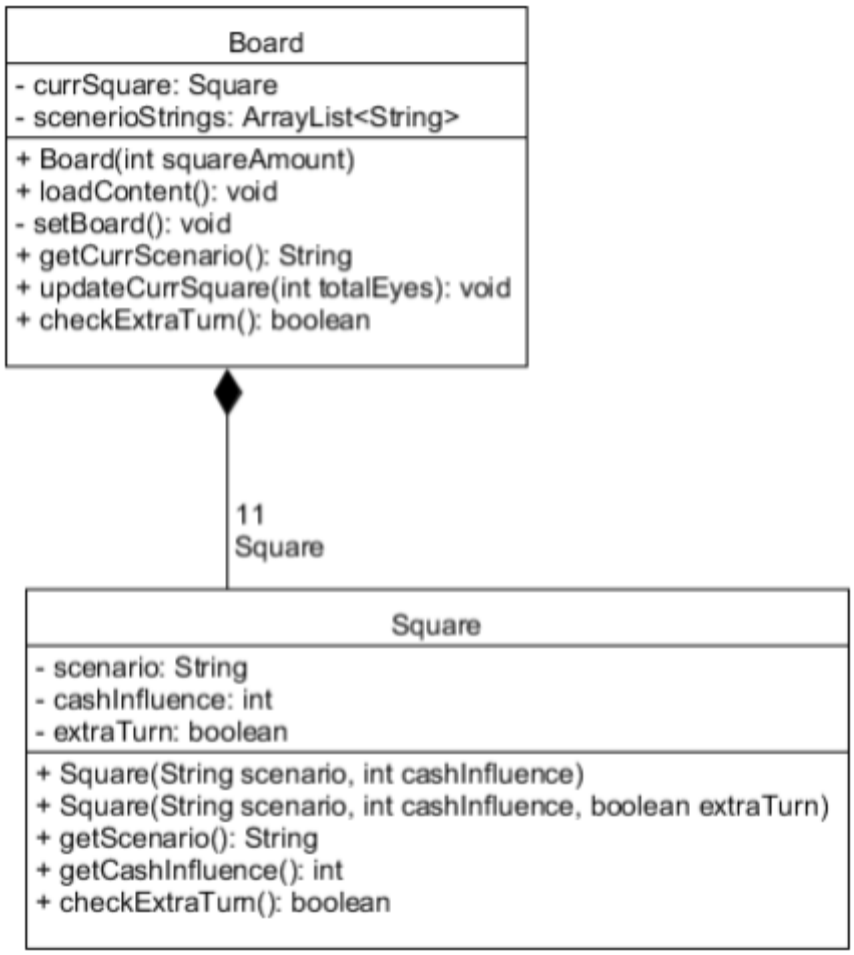
Figur 8: Lagdeling set i Design Klasse Diagram

Første klasse der kan skille sig ud er GameTool der ligger nederst i diagrammet. Dette er en unik klasse i forhold til de andre, i det at den aldrig skal instantieres, men dens metoder skal bruges. Den bliver dog stadig brugt af dele af systemet, som vist ved de stiplede pile til klassen.



Figur 9: "GameTool" klasse, set i Design Klasse Diagram

I associationen mellem Board og Square ses en ‘composite aggregation’. Dette betyder, at Board står for at oprette instanser af Square og indeholde dem. Derved vil det også få betydningen, at hvis instansen af Board bliver slettet, vil alle dens referencer til instanser af Square gå tabt.



Figur 10: Board og Square klasser med association, set i Design Klasse Diagram

Beskrivelse af anvendte GRASP-mønstre

Under projektet er GRASP principperne benyttet, til at tildele ansvar mellem spillets software klasser. Et overordnet formål med dette er, at gøre klasserne mere overskuelige, genbrugelige og vedligeholdelsesvenlige. Dette bliver gjort ved at analysere klassernes “gør” ansvarligheder og “kende” ansvarligheder. I dette afsnit gives et par korte eksempler på, hvor nogle af GRASP principperne er blevet brugt i systemet, og om hvordan end det har indflydelse på hele systemets design og konstruktion.

Et af GRASP principperne er “Information Expert”, som handler om at give ansvaret for funktioner til den klasse med nok information til at udføre opgaven. Dette kan f.eks. afspejles i klassen “Die”. Klassen skal symbolisere en terning i den virkelige verden, og kender derfor til 3 int værdier i form af den mindste, den højeste og det aktuelle (daværende) antal af øjne. Da klassen kender til disse informationer, giver det mening, at den får ansvaret for at agere med informationen, ved at kaste med terningerne, selvom man i den virkelige verden kunne argumentere for, at det er et menneske (f.eks. “Player” klassen), der kaster terningerne. Så selvom det er spilleren (klassen “Player”), der bruger dem i spillet (klassen “Game”), så, fordi informationen “opbevares af” terningen (klassen “Die”) er det den, der har ansvaret for det. Derimod for at samle værdien af ens terninger, skal man kigge på hvilken klasse, der indeholder terningerne, og derfor kender til dem. Dette gør klassen “Game”, og det er derfor dens ansvar at lægge de to terningekast af de enkelte terninger sammen, for at få selve kastets værdi. Ansvaret her er tydeligt fordelt efter hver klasses “kende” ansvarligheder.

Denne fremgangsmåde understøtter også to andre GRASP principper, som går ud på at få lav coupling[4] og høj cohesion[5]. Ses der videre på “Die” klassen, har den altså lav coupling, da den ikke skal bruge relationer til andre klasser, for at udfører dens ‘kast terning’ metode (rollDie()). Cohesion kan derimod sættes til at være høj i dette eksempel, da det at kaste en terning, og en terning har stor sammenhæng, resulterer i en fokuseret og forståelig klasse.

Princippet om “Creator” er blevet taget i brug, hvilket kan ses under oprettelsen af “Board” og “Square”. Da der ikke er meget brætspil over et bræt uden felter på, giver det mening, at brættet (klassen “Board”) har ansvaret for at ‘oprette’ og ‘indeholde’ felterne (instanser af klassen “Square”). Dette bliver vist i koden ved, at det er “Board”, der instantierer felterne, og derefter holder dem i et array. Her har klassen (“Board”) altså både en “gør” ansvarlighed og en “kende” ansvarlighed, hvilket opfylder kravet for at være en “Creator”. Hertil kan det tilføjes, at “Creator” er en form for “Information Expert”, men et lidt mere specifikt begreb bliver brugt da den ‘opretter’. Dette giver lav coupling, da det sætter programmet op med færre unødvendige associationer mellem klasserne, for oprettelse af elementer.

Til sidst kan “Controller” princippet vurderes brugt, da dette er brugt under oprettelsen af klassen “Game”. Dette princip introducerer controller-laget til koden, i stedet for en simpel 2-laget model med kun UI og domæne lag. Her kan der oprettes klasser for at få kommunikationen med kunden (fra UI-laget) håndteret med informationen i systemet (fra domæne-laget). Idet spillet er ret simpelt, og der ikke har været nogen hentydning fra kunden, i kravene, om at udvide det, er der kun blevet oprettet én klasse, klassen “Game”, til at koordinere arbejdsopgaverne i programmet mellem bruger og systemet. Dette bidrager også til lavere coupling for programmet.

[4] Se ordbog - coupling
[5] Se ordbog - cohesion

Implementering

Beskrivelse af branching strategi

Vi har opdelt vores projekt i flere “branches” med formålet at gøre udviklingen mere struktureret, undgå unødvendig merge konflikter og gøre det muligt at udvikle flere komponenter samtidigt, uden at disse påvirker hinanden. Nedenfor beskrives der “branches”, som blev lavet til dette formål.

Projektets “repository”, som er under navnet CDIO, har/havde disse “branches”:

- del2_master (default/base), hvor kun publiceres fungerende og testet kode, altså klar til brugen. Commits er ulovligt i denne branche, kun merge er tilladt.
- del2_release, hvor der laves de sidste ændringer, før koden bliver publiceret til del2_master
- del2_developing, hvor der udvikles spillets logikken og UI.
- del2_readFromFileFeature, hvor der udvikles en service til at læse fra en tekstfil.
- del_1 er en branch som indeholder koden fra CDIO del_1 version.

Kodning

Generelt har koden for det fungerende spil, været meget som det forrige program. Vi har valgt at fremhæve en del af koden, som indeholder noget nyt vi har lært, og er en af de mest avancerede aspekter i programmet.

readFromFile() i GameTools

```
public static ArrayList<String> readFromFile(String filename) throws IOException{
    ArrayList<String> values = new ArrayList<>();  // declaring arraylist with name "va
    values.add("spacing.....");   // add an element to 0 index, so element in index 1 i
                                  // in the list in line number 1. It makes just easier
    String file ="languages"+ File.separator +filename+".txt"; // inserting filename pa

    BufferedReader reader = new BufferedReader(new FileReader(file));  // creates reade
    String currentLine = reader.readLine();  // reads a line
    while (currentLine != null){ // runs loop until the current line is not empty.
        values.add(currentLine);   // add line as an element to the array list
        currentLine = reader.readLine();  // reads next line
    }
    reader.close();

    return values;
}
```

Kodeklip 1: readFromFile() metode (Linje 42-56 i GameTool klassen)

En af de mere interessante aspekter af programmet er kommet via brugen af 2 Reader klasser fra java.io bibliotekket, BufferedReader og FileReader. Dette bliver brugt til at læse fra et tekstdokument, der ligger udenfor selve programmet, og bliver brugt til at sætte UI og scenarierne for spillet op. Dette kan gøre det nemmere at skifte sprog for videre udvikling af programmet.

Metoden vi er kommet frem til virker overordnet god, med et lille twist til at indsætte en String, som vi har valgt indeholder ordet “spacing…..”. Denne ‘spacing’ gør at linje numrene på dokumentet, der bliver læst er identiske til placeringen i den ArrayList de bliver lagt i, for nemmere relation. Ved at bruge *File.separator* bliver der her undgået et problem, der kan opstå med hvordan Mac og Windows skriver deres filepaths på, da Mac, der bruger “/”, hvor at Windows bruger “\”.

Tests

Udførsel	**Forventet**	**Aktuel**	**Pass/Fail**
Test case [TC3]: accountNegativeTotalCashTest()			
Udfører oprettelse af en 'spiller-konto', og tester om denne er oprettet med det rigtige start beløb. Prøver derefter at indsætte et negativt beløb mindre end selve saldoen (testes), fulgt af et beløb, der gør at saldoen skal ramme nul (testes). Til sidst indsættes et negativt beløb på kontoen, som jo skal forblive nul, og ikke må gå i negativ saldo (testes).	Kontoens saldo vil stoppe ved nul, og vil derfor ikke gå i negativ ved indsættelse af negativ saldo større end balancen. Yderlige vil registreringer af negative indsættelser på saldoen, når kontoen er nul, heller ikke trække i negativ.	Kontoens saldo bliver ikke mindre end nul.	Pass

```
void accountNegativeTotalCashTest() {
    Player testPlayer = new Player( number: "testNegativeCash");
    int startingCash = testPlayer.getTotalCash();

    testPlayer.addToCash( cashInfluence: -999);
    int cashAfterNegativeInput = testPlayer.getTotalCash();
    testPlayer.addToCash( cashInfluence: -1);
    int cashAtZero = testPlayer.getTotalCash();
    testPlayer.addToCash( cashInfluence: -10);
    int cashAfterNegativeValue = testPlayer.getTotalCash();


    //Checks if the Player's Account's getTotalCash starts with 1000, as it is re
    assertEquals( expected: 1000, startingCash);

    //Checks if addToCash works when a negative cashInfluence is added and the Pl
    assertEquals( expected: 1, cashAfterNegativeInput);

    //Checks if addToCash works when adding a cashInfluence with the negative val
    assertEquals( expected: 0, cashAtZero);

    //Checks if addToCash stops at 0, as is requested, when adding a negative cas
    // make the Player's Account's totalCash go into the negative had no precauti
    assertEquals( expected: 0, cashAfterNegativeValue);
}
```

Udførsel	**Forventet**	**Aktuel**	**Pass/Fail**
Test case [TC4]: winnerFoundTest()			
Opretter en testversion af spillet med én spiller i, spilleren er oprettet som normalt. Herefter bliver hans saldo sat til 1 under hvad han skal bruge for at vinde. Efter der bliver lavet kontrol på om spilleren har vundet på det punkt, bliver der tilføjet 1 til hans saldo og der bliver så kontrolleret om han har vundet. Derefter bliver der igen tilføjet 1 for at genkontrollere om spilleren stadig har vundet i denne tilstand.	Første gang skal spilleren ikke have vundet. Ved de to næste kontrolpunkter skal spilleren have vundet.	Spilleren vinder først når han har ramt 3000, men har også vundet hvis han er over 3000.	Pass

```
void winnerFoundTest() {

    // Arrange
    Player pl = new Player( number: "test player");
    pl.addToCash( cashInfluence: 1999);
    Game testGame = new Game(pl);  // current player: 1

    // Act
    boolean isPlayer1WinnerWith2999 = testGame.winnerFound();
    pl.addToCash( cashInfluence: 1); // sets player cash to 3000
    boolean isPlayer1WinnerWith3000 = testGame.winnerFound();
    pl.addToCash( cashInfluence: 1); // sets player cash to 3001
    boolean isPlayer1WinnerWith3001 = testGame.winnerFound();

    // Assert
    assertFalse(isPlayer1WinnerWith2999);
    assertTrue(isPlayer1WinnerWith3000);
    assertTrue(isPlayer1WinnerWith3001);
}
```

Konfiguration

Systemkrav

Hardware krav[6]

2 GB RAM minimum, 4 GB RAM recommended
1.5 GB hard disk space + at least 1 GB for caches
1024x768 minimum screen resolution

Krav til styresystemmet[7]

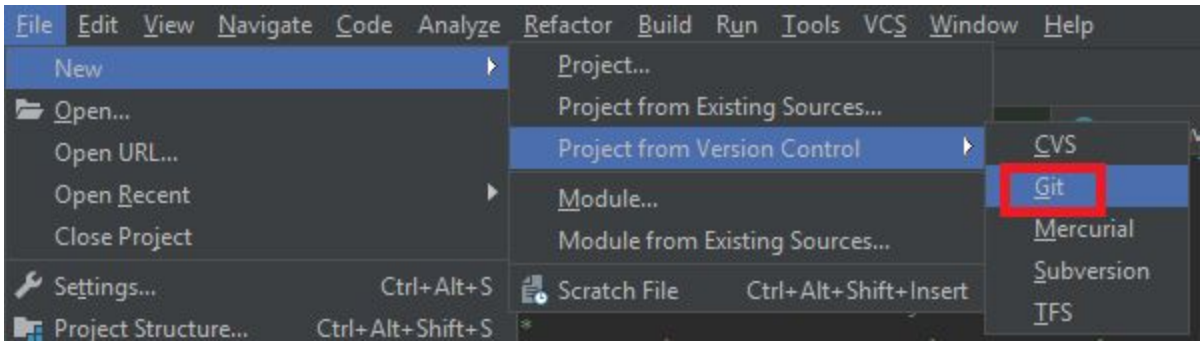
- Windows
    - Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)

Software

- JDK version 1.8.

Importering projektet fra et git repository

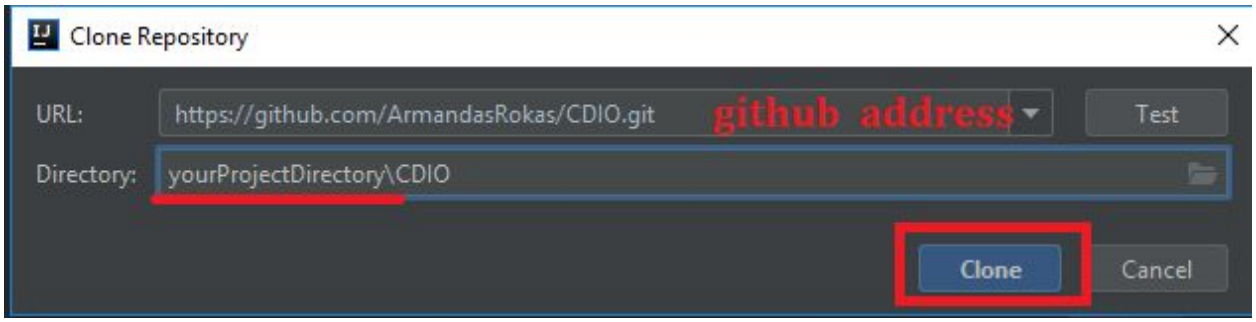
1. Tryk på New -> Project from Version Control -> Git



2. Skriv en address http://github.com/ArmandasRokas/CDIO.git under URL.
3. Ændre “yourProjectDirectory” til en placering, hvor du vil have projektet på din computer.

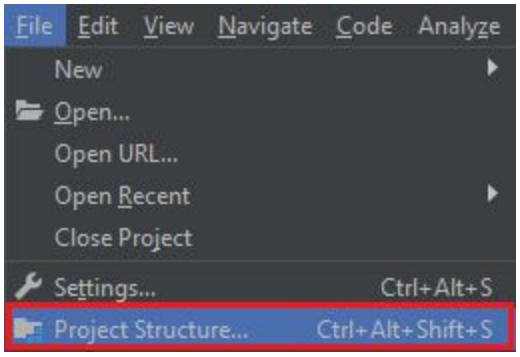
[6] https://www.jetbrains.com/idea/download/#section=windows
[7] https://www.jetbrains.com/idea/download/#section=windows

4. Tryk på Clone.

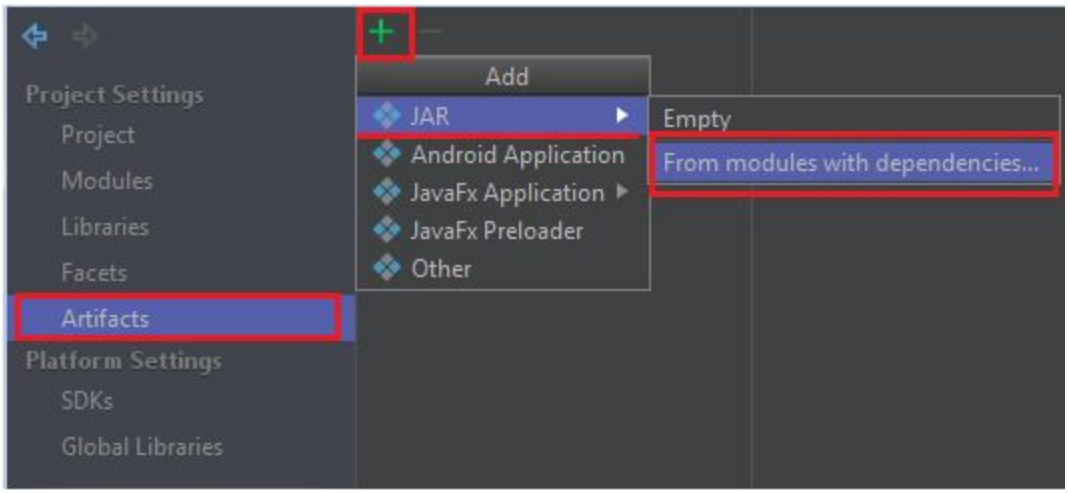


Vejledning i hvordan kildekoden compiles, installeres og afvikles

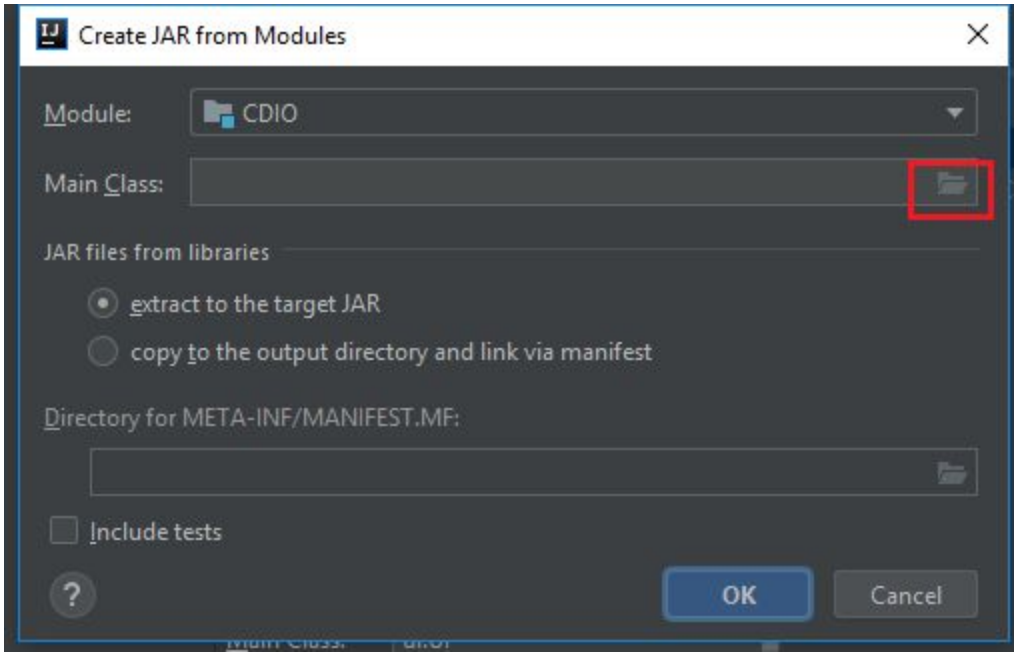
1. Tryk File -> Project Structure



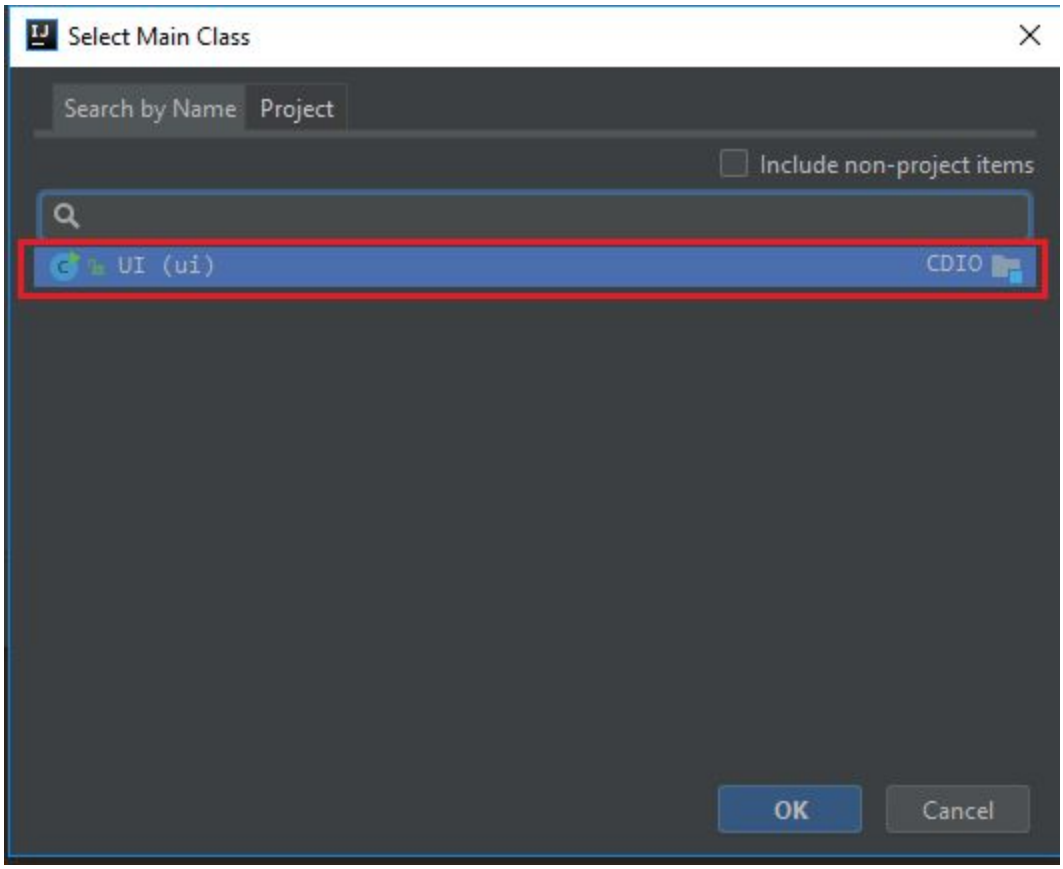
2. Tryk Artifacts -> Add (som er markeret med et grønt plustegn) ->JAR->From modules with dependencies.



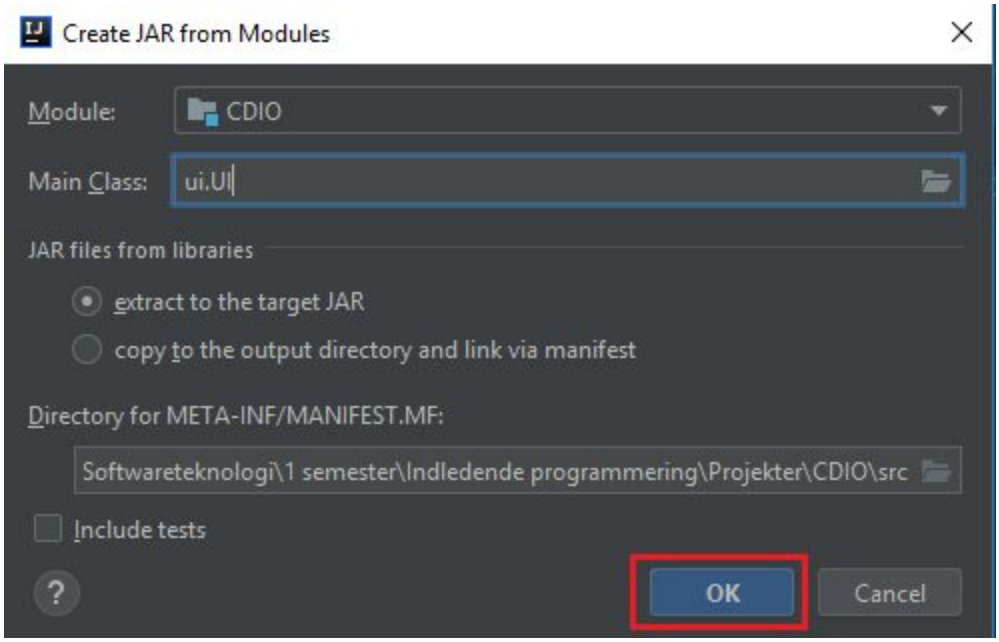
3. Tryk på “mappen”



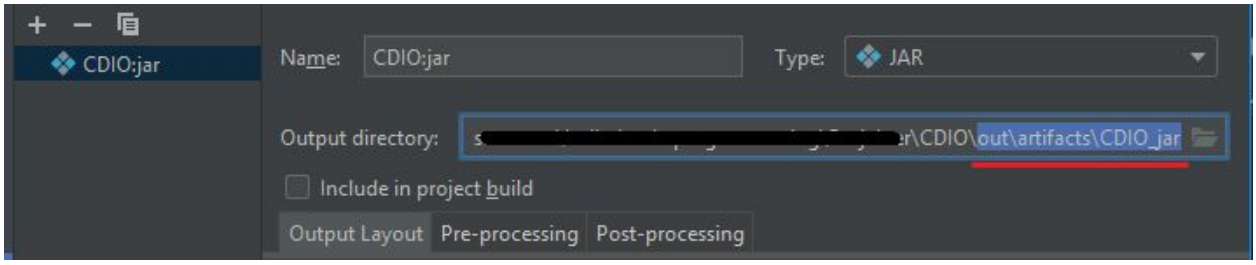
4. Vælg Main class, som er UI.



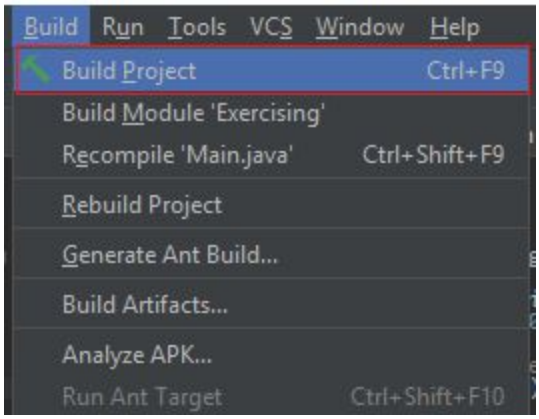
5. Bekræft med OK



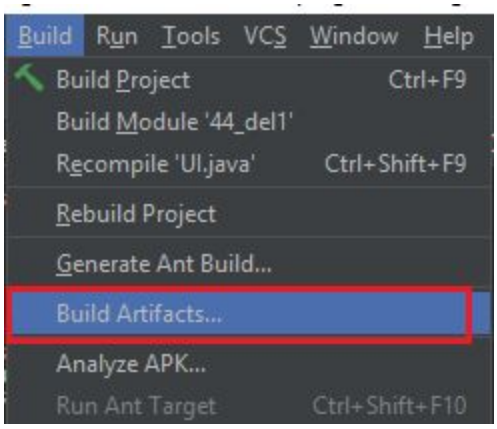
6. Slette out\artifacts\CDIO_jar fra Output directory



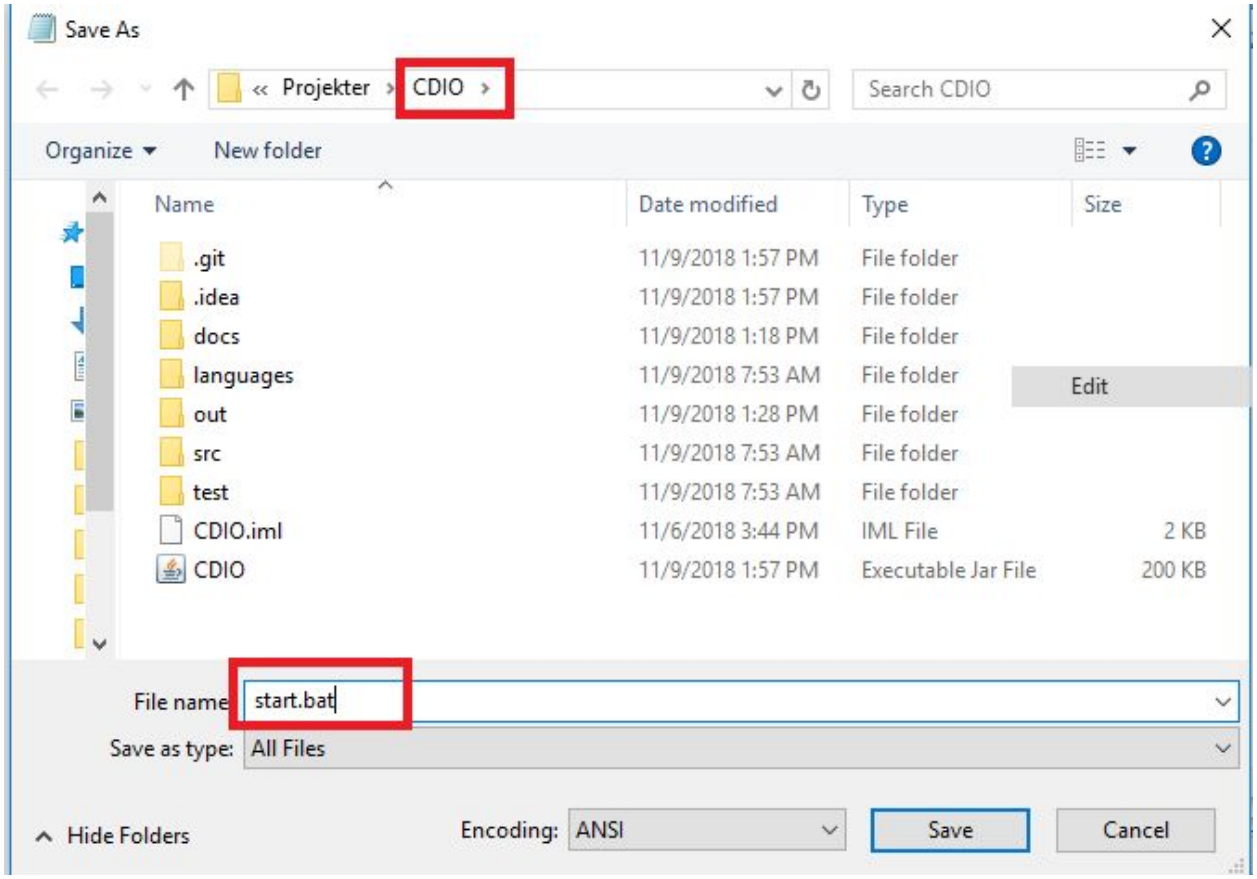
7. Tryk på Build -> Build Project



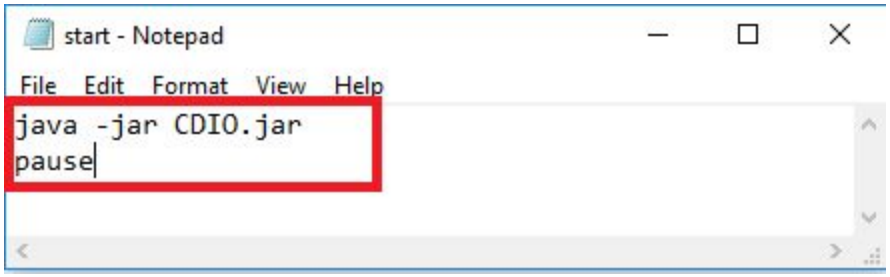
8. Tryk på Build -> Build Artifacts



9. Gå til mappen yourProjectDirectory\CDIO
10. Lav en ny start.bat fil i notepad



10. Tilføj “java -jar CDIO.jar pause” til start.bat filen.



11.Tryk på start.bat filen, som lige blev lavet, til at starte spillet

Github release
Det er også muligt at finde den nyeste binære fil i github under adressen:
https://github.com/ArmandasRokas/CDIO/releases

Projektforløb

Branching og synkron kodning - refleksion
Dette har været vores første større projekt hvor flere har arbejdet sammen over branches. Alt i alt er det gået godt, men på et tidspunkt kom flere til at arbejde i én branch. Det blev fikset, men tog tid, der kunne være blevet brugt bedre i projektet. Gruppen har prøvet at finde frem til problemet og har fundet en teori, dette vil der blive læst op på og om muligt testet omkring. Til fremtiden vil flere under-branches højst sandsynligvis blive sat i brug som en start på løsningen.
Introduktion på projektforløbet
Vi har valgt at forsøge at bruge “Unified Process” modellen til at gennemføre projektet. Dette betyder, at vi har opdelt projektforløbet i fire stor faser, dvs. Inception, Elaboration, Construction og Transition, og seks “sprints” på tre-fire dag længde, som kan ses i tabellen nedenfor. I hvert sprint skal vi så meget som muligt gennem analyse, design, implementation og test disciplinerne. Dette er både for at sikre, at vores krav bliver opdateret i forhold til feedbacks fra kunden og teste, og fordi vi vil have en fungerende og testet kode i slutning af hvert sprint.

Inception	Elaboration		Construction		Transition
Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6
22/10-24/10	25/10-28/10	29/10-31/10	01/11-04/11	05/11-07/11	08/11-09/11

Refleksion på projektforløbet

Det lykkedes ikke helt at implementere Unified Process. En af grundene kunne være, at projektet var alt for lille, hvilket gør det svært at inddele iterationer og fordele arbejde ligeligt. Dette resulterede i, at der blev arbejdet med en proces, der måske i lidt større grad ligner Waterfall.

Projektet forløb dog uden markante problemer, og blev ikke mødt af betydelige ”vejbump”. Der blev diskuteret og besluttet ud fra faglige synspunkter ved mange lejligheder, og vi mener, at vi endte ud med en god kommunikation gennem rapporten. Der er også givet illustrationer i form af UML-artefakter[8], for at give en overskuelig forståelse primært til os selv, men også læser.

Vi brugte dog ikke en *toString*-metode, som det var et krav. Det gav heller ikke mening for os, eftersom vi først indså manglen, når vi havde løst det på en anden måde. Vi benyttede altså andre værktøjer (*getters*) til at udføre det eneste formål, hvor vi mente, at der potentielt kunne have være et formål ved at bruge en *toString*-metode.

Konklusion

Ved analyse af kundens krav er der opstillet et virkeligheds-koncept af det simple matador spil, for at understøtte en generel forståelse. Denne generelle forståelse ledte videre til et mere konkret idé-design af spillet. Efterfølgende, blev dette idé-design grundlag for en implementering af selve spillet, som program. Yderligere, er der i rapporten givet fordybelse til opbygningsmønstre for, hvordan dele af programmeringen ideelt set, har indflydelse på hinanden.

Dette projekt kan vurderes til at være en succes, da der er gennemført et samlet og lærerigt læringsforløb, samt udviklet et færdigt program med følgende dokumentation. Hvis der skulle være en mangel for at succesbetegnelsen blev opfyldt, skulle det være, at der mangler en implementering af en *toString*-metode, hvilket var et krav, som ikke blev opfyldt. Dette blev dog løst på en anden måde, og slutresultatet fungerer.

[8] UML-artefakter - Se ordbog

Ordbog

Aktør	En, der har en direkte rolle i et samspil med det aktuelle system.
Loop	En gentagelse af noget (hvor *noget* ofte er markeret eller afgrænset i en ramme eller lign.).
Sprint	En arbejdsperiode i projektforløbet, hvor der så meget som muligt praktiseres analysis, design, implementation, test disciplinerne.
Coupling	Coupling omhandler hvor mange andre klasser en klasse refererer til eller er afhængig af. Det er foretrukket at have lav coupling. Går hånd i hånd med *cohesion*.
Cohesion	Cohesion derimod omhandler klassens ansvar, om dette er tydeligt og at den ikke har for meget og derfor kunne ende med at blive uoverskuelig. Det er foretrukket at have høj cohesion. Går hånd i hånd med *coupling*.
Lag (når man snakker kode)	For at gøre koden overskuelig kan det være praktisk at inddele den i lag. Dette kan f.eks. være en 3- eller 4-lags struktur og opdeles i f.eks. ui, controller og domæne. Her står ui for kommunikation af brugeren, controller for logikken og domæne for informationerne.
int	'int' er en forkortelse indenfor programmering af 'Integer' og betyder 'heltal' på engelsk.
UML-artefakter	En beskrivende klassificering, der beskriver en enhed eller en måde at se information på, under et softwareprojekt. Kan være på både illustrationer og tekstform. (UML er Unified Modeling Language)
UI	User Interface. En brugerflade som er det, der er i direkte kontakt med aktøren.

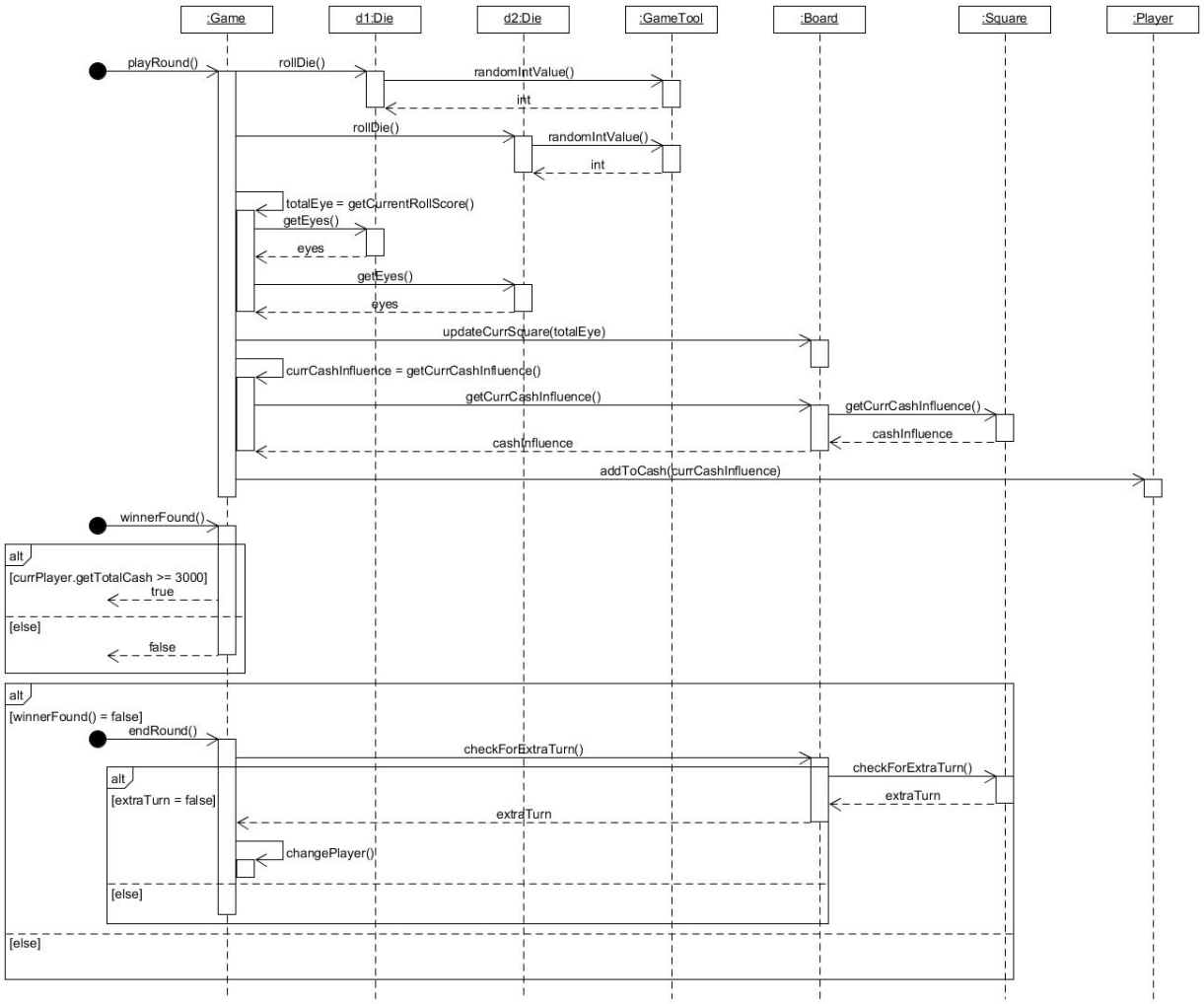
Kildeliste

https://www.nngroup.com/articles/response-times-3-important-limits/
Medie: artikel - Udgivelsesår: 2013 - Forfatter: Expert
Vurdering: *Ikke meget pålideligt* - Den benyttede oplysning om 'response time' er givet fra en pålidelig kilde, men kan være ændret ud fra nyere undersøgelser, da dette er fra 1993. Oplysningen af denne kilde er dog godkendt af kunden (lærer), hvilket giver grundlag for vores benyttelse der af.

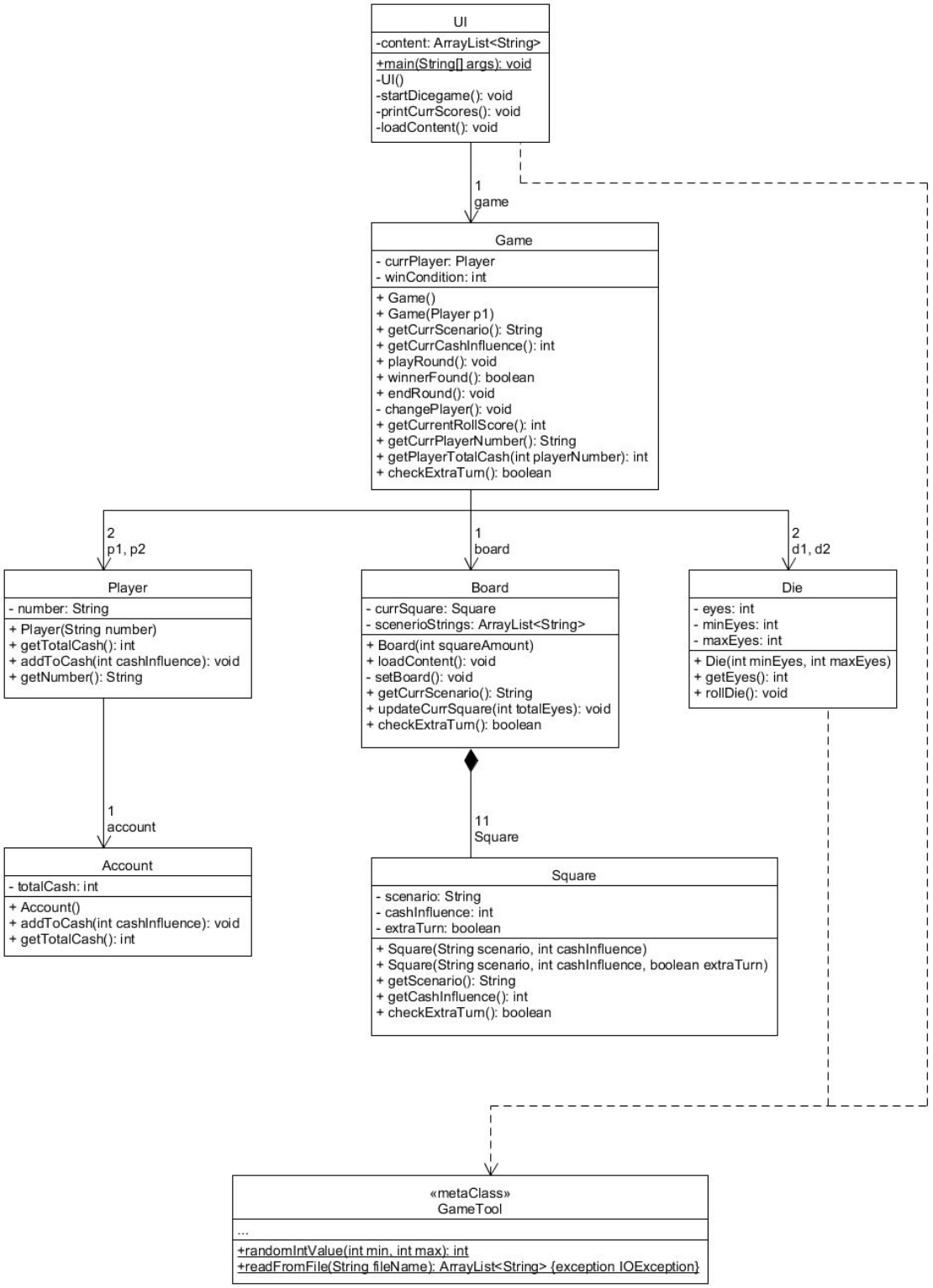
https://www.jetbrains.com/idea/download/#section=windows
Medie: hjemmeside- Udgivelsesår: ukendt- Forfatter: ukendt
Vurdering: *pålideligt* - almindeligt anerkendt programmerings info-hjemmeside.

Bilag

Bilag 1: Fuld Sekvensdiagram af usecase “Spil spillet”



Bilag 2: Fuld Design Klasse Diagram af programmet



Bilag 3: Gantt kort

2018 dato	man 22/10	fre 26/10	man 29/10	man 5/11	tors 8/11	dag I uge 45	afl. fre 9/11
Startvurdering							
Kravspecifikationer							
Use case(s)							
Modellering og beskrivelse							
Design til implementeringshjælp							
Implementering							
Test-implementering							
Rapport skrivning							
Slutrettelser og færdigskrivning							
	Planlagt						
	Kan ændres						
	Kan laves som lektie til næste gang						

Evaluering af gantt kort tids-tilpasning: Vi har ikke brugt dette gantt kort så meget, og vi har mødtes lidt flere gange end det er blev planlagt. Dog passede opgavefordelingen rimelig godt med datoerne, udover lidt ekstra rettelser og forbedringer på afleveringsdagen.

Bilag 4: Liste over primære ansvarsområder

Liste over primære ansvarsområder						**Status**
Opgave	David	Emil	Kristian	Simon	Armandas	
Vision + aktør + SS +					**x**	
Fully dressed: Spil spillet i forhold til Kristians workflow diagram				**x**		
Domænmodel					**x**	
SSD		**x**				
SD			**x**			
Package diagram				**x**		
Designklassediagrammet torsdag			**x**			

tilrette getRandomNumber test, at den passer til den nye rollmetode.		x				
Implementation af GUI	x					
skrive en test for winnerFound					x	
Beskrivelse af GRASP					x	
Lav en test der sandsynliggør at balancen aldrig kan blive negativ, uanset hvad hæv og indsæt metoderne bliver kaldt med. **test case beskrives**		x				
Beskrive kode			x			
indledning og abstract		x				
Kommenter i koden og finskrivning			x		x	
Konklusion		x				
rette UI	x				x	
ordbog	x	x	x	x	x	
beskrive Branching strategi			x		x	
beskrive test case: winnerFound()			x			
konfiguration					x	
release branch and master						
code coverage						
brugervenlighedstest						


Færdig
Skal bekræftes
Startet
Ikke startet
Ikke obligatorisk