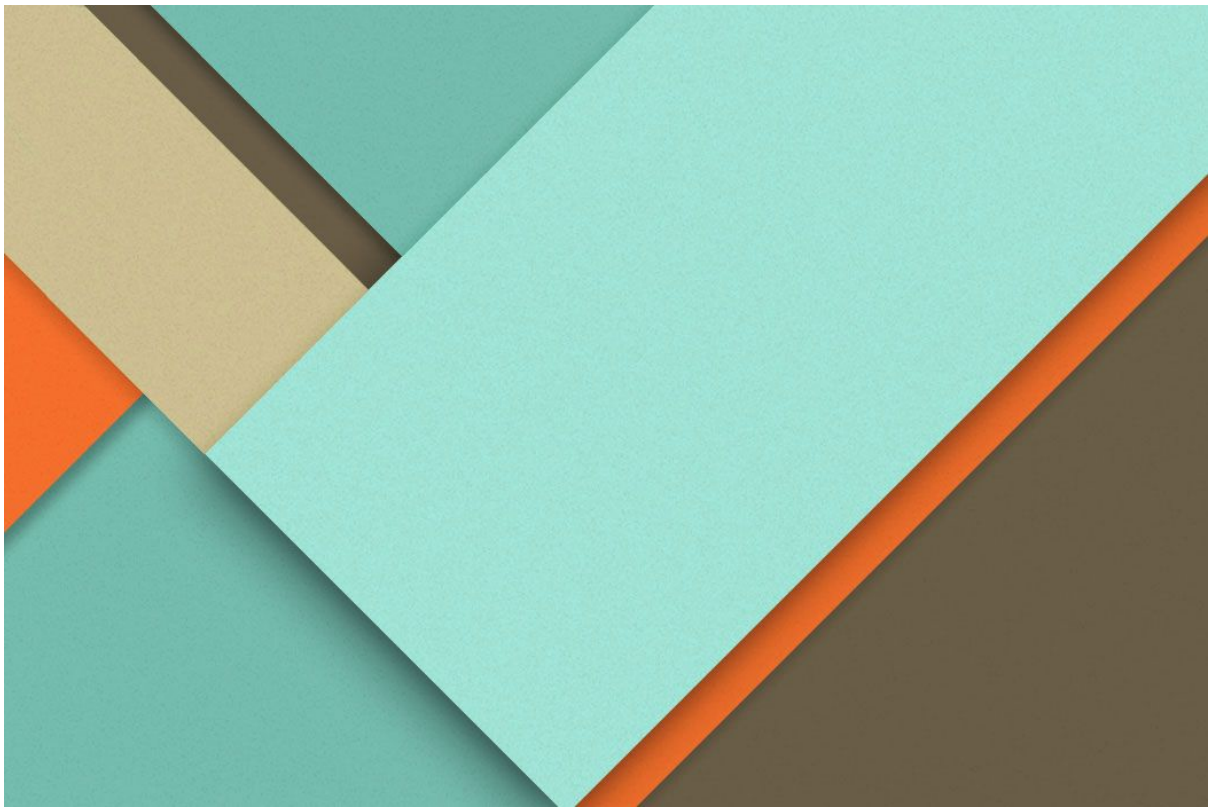


CDIO delopgave 3

Kursus nr. 02313 - 02314 - 02315

Alexander Kjeldsen (s165477)

Elísa Kristín Sverrisdóttir (s175115)

Josephine Mellin (s163313)

Mads Jørgensen (s175184)

Oliver Uldall Schultz (s175113)

Gruppenr. 15

Afleveringsdato: 1. december 2017

Time regnskab

Deltager	Design	Impl.	Test	Dok.	Andet	I ALT
Alexander	2	7	5	4	2	20
Elísa	2	2	0	10	6	20
Josephine	5	5	1	7	2	20
Mads	4	11	1	4	1	21
Oliver	7	6	1	4	2	20
Sum						**101**

Abstract

Denne rapport omhandler vores CDIO del 3 projekt. Denne gang var formålet at lave et program, der simulerer et Monopoly Junior spil. Spillets regler er beskrevet i detaljer i vores kravanalyse og i vores fully dressed beskrivelse af vores use case “PlayGame”.
Programmet udvikles i java, og vi gør brug af en GUI. Vi har beskrevet programmet og dets design i mange forskellige diagrammer og artifacts, som er nemt forståelige.
Vores artifacts og diagrammer giver et godt billede af, hvordan selve koden er opbygget.
Rapporten indeholder også en beskrivelse af implementering og testning samt en beskrivelse af vores arbejdsprocess, som er beskrevet via unified process. Vi har opfyldt kundens ønsker om forskellige artifacts og relevant dokumentation.

Indholdsfortegnelse

1.Indledning

I denne opgave får vi, IOOuterActives, som programudviklere til opgave at udvikle et Monopoly Junior spil. Der angives flere regler, som vi tager højde for. Som i vores tidligere rapporter har vi valgt at stille vores krav op efter FURPS-metoden, da det giver et godt overblik over Christians, vores kunde, krav. Vi gør brug af forskellige artifacts og diagrammer til at beskrive, hvordan spillet fungerer.

2. Analyse

Vi har valgt at bruge FURPS for at beskrive kundens krav, hvori der bådes beskrives de funktionelle krav, F eller 'Functional requirements', men videre også *ikke*-funktionelle krav, som indebærer U, R, P og S, som henholdsvis står for 'Usability', 'Reliability', 'Performance' og 'Sustainability'[1].

FURPS

- **Functional:**

F1: Vores kunde ønsker et spil, der kan spilles af 2-4 personer.
F2: Kunden ønsker at spillet skal bestå af en spilleplade med flere forskellige felter.
F3: Når man lander på et felt, skal man som krav kunne rykke videre derfra på det følgende slag med terningerne.
F4: Hver spiller har en brik (en bil), som har hver sin farve.
F5: Spilles spillet af to, starter de hver især med en pengebeholdning på 20 dollars. Spilles det af tre får de hver 18 dollars. Hvis spillet derimod består af 4 deltagere starter de med 16 dollars.
F6: Hver spiller har mulighed for at købe de felter, som ikke er specialfelter som 'Go to prison', 'Dolphin show', 'Free parking', 'Prison visit' og 'Start'. Når et felt købes, indrammes det er samme farve, som spillerens brik har.
F7: En spiller rykker det samme antal felter, som terningens øjne viser.
F8: Rammer en spiller et felt som er opkøbt, opkræves der et beløb (taget fra den givne spillers pengebeholdning), som gives til den spiller, der er ejermanden af feltet.
F9: Et felt kan ikke opkøbes af to forskellige spillere.
F10: Har en spiller to felter, som er opkøbt, skal en given spiller, som lander på modstanderens felt betale et dobbelt beløb til ejermanden.
F11: Lander en spiller på et 'Go to prison'-felt, må han/hun sidde en tur over og betale 1$ til banken.
F12: Lander en spiller på et 'Dolphin show'-felt, skal den givne spiller betale 2$ til banken.
F13: Hver gang en spiller kommer over feltet 'Start' gives han/hun 2$.
F14: Hvis en spiller rammer feltet med et tog, må han/hun slår en ekstra tur.
F15: Spillet slutter, så snart én spiller er gået bankerot, hvorefter der findes en vinder ved den spiller, som har flest penge tilbage i sin pengebeholdning.

[1] Larman, 2005, side 57

F16: Vi har valgt at nedprioritere kravet om chancefelter. Disse felter et indtil videre erstattet med 'Train' og 'Dolphin show'-felter.

- **Usability:**

U1: Spilleren skal kunne se, den værdi terningen slår.
U2: Spilleren skal kunne se sin pengebeholdning.
U3: Spillepladen skal være brugervenlig og tilpasset den målgruppe, som skal kunne spille spillet: Børn.
U5: Spillerne skal nemt kunne se, hvis tur det er, og hvor deres bilbrik befinder sig, og hvilke felter, der er opkøbt.
U4: Spillets responstid skal være kort.
U5: Der skal indgå en brik for hver af de spillere, som spiller spillet, som vises på spillepladen.
U6: Programmet skal meddele, når man enten mister eller tjener penge til en pengebeholdning.
U7: Programmet skal meddele, når en spiller er fundet, og spillet videre er slut.

- **Reliability:**

R1: Kunden har ikke angivet konkrete krav af responstiden ift terningekast og ryk af brikker, men vi har valgt at sætte den til 333 millisekunder.
R2: Det forventes at programmet kan aflæse et vilkårligt felts funktioner.
R3: Det forventes at hver spiller starter med en pengebeholdning på på 20 dollars, hvis de er to spillere. 18 dollars, hvis de er tre og 16 dollars, hvis de er fire.
R3: Det forventes at programmet via feltaflæsningen kan enten trække fra eller lægge penge til, når en spiller lander på en modstanders købte felt.
R4: Systemet skal kunne udprinte diverse meddelelser, som en tilpassede de forskellige felter og deres funktioner.
R5: Det forventes videre at spillet fortsætter, indtil at én af spillerne går bankerot, og videre at programmet, kan aflæse de resterende spilleres pengebeholdning og angive en vinder.

- **Performance:**

P1: Terningens øjne vises efter 333 millisekunder.
P2: Brikkerne skal rykkes efter terningens øjne er vist.
P3: Videre skal brikken rykkes det antal felter, som er lignende antal af terningens øjne.
P3: De meddelelser, som spillet skal give ift. de tilsvarende felter, skal ligeledes gives senest efter 333 millisekunder.

- **Supportability**

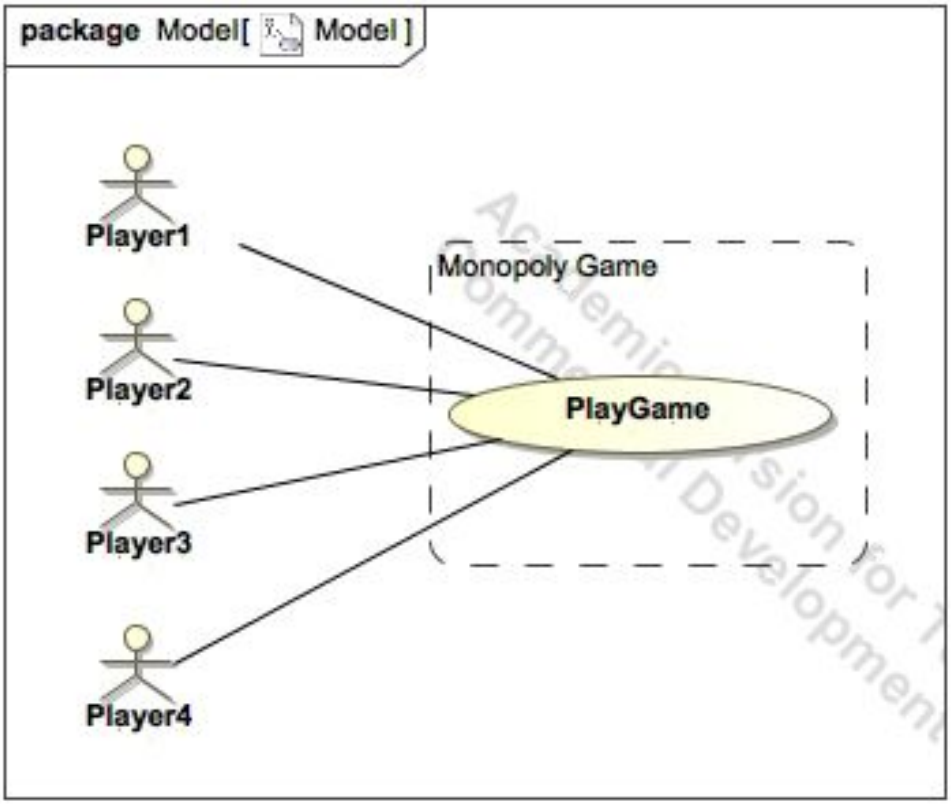
S1: Spillet skal kunne spilles på DTU's databarer, som kører windows 10.

Vi har valgt at se bort fra kravet, om at den yngste starter, da vi ønsker at spillets deltagere hurtigt skal kunne starte spillet. I stedet har vi valgt at spillet udprinter en besked til spillerne, om at det er den yngste der starter.

Use Case description
ID 1: PlayGame
Brief
2-4 spillere (Player1, Player2, Player3, Player4) med hver sin forskelligfarvet bilbrik, starter med en pengebeholdning på henholdsvis 20, 18 og 16 dollars.
Spillet foregår på en spilleplade med 24 felter, der både indeholder felter, der har forskellige funktioner, og hvor nogle kan opkøbes. Rammer en modstander et ejet felt, skal han/hun betale til ejermanden af feltet. Det betalte beløb fordobles, hvis modstanderen er indehaver af to felter med samme farve.
Spillet fortsætter indtil at én af spillerne går bankerot, hvor det derefter er den spiller, som har flest penge tilbage vinder.

ID 1: PlayGame
Casual
- De primære aktører (Player1-4) starter spillet ved at skrive antal af spillere ind i systemet, og vælger herefter hver sin farve.
- Hver især starter de med en pengebeholdning på en forskellige sats alt efter, hvor mange spillere, der deltager.
- Er de to, starter de hver med 20 dollars, er de tre starter de med 18 dollars og er de fire, starter de med 16 dollars.
- Det er herefter den yngste af spillerne, som starter.
- Den givne spiller slå med terningens og rykker det samme antal felter, som terningernes øjnene viser.
- Rammer spilleren et felt, som han/hun gerne vil købe, betales banken samme beløb, som er angivet på feltet. Feltet bliver herefter indrammet med samme farve, som er lignende den givne spillers brik.
- Rammer en anden spiller et købt felt, skal han/hun betale “husleje” til ejeren
- Har en spiller købt to felter af samme farve, skal den eventuelle modstander, som lander på et af de felt, betale en fordoblet “husleje” til ejeren.
- Rammer en spiller et ‘Train’-felt får den givne spiller en ekstra tur.
- Rammer en spiller et ‘Go to prison’-felt, skal denne spille sidde en tur over og betale 1$ til banken.
- Rammer en spiller et ‘Dolphin show’-felt, skal den givne spiller betale 2$ til banken.
- Alle spillere gives 2$, hver gang de krydser ‘Start’-feltet.
- Spillet fortsætter indtil én af spillerne går bankerot, hvor den af de øvrige spillere, med fleste penge tilbage i sin pengebeholdning udnævnes vinder.

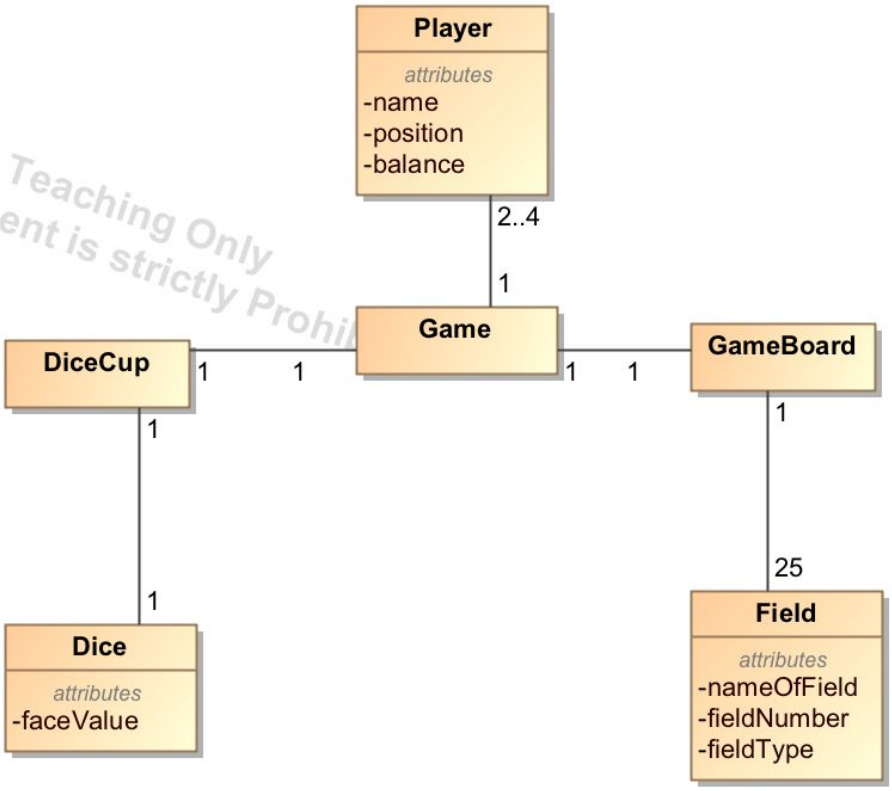
Use-Case diagram:
Gennem vores use case diagram, forsøger vi at skabe et billede over systemet og den aktør. Som i dette tilfælde er PlayGame (use case) Player1-Player4 (primær aktør). Det foretrækkes at use case diagrammet er simpelt og at man bruger use case beskrivelsen til at gå i detaljer.[2]



Use-case diagram lavet i MagicDraw

[2] Larman, 2005, side 158

Domæne Model:
En domæne model (domain model) illustrerer vigtige begreber i et domæne.
For at nedbryde og overskueliggøre domænet, må man identificere koncepterne, attributterne og forbindelser, der er relevante. Her gives der blot et billede af koncepterne der hører hjemme i "den virkelige verden", og ikke af selve software objekterne. Dog kan den kan være en nyttig inspirationskilde for software designet og navngivning af klasser i designfasen[3].



Domain model lavet i MagicDraw

Use Case Specification: PlayGame
ID: 1
Brief description: The game requires at least 2 players and max 4. Each player receive an amount of dollars before the game starts, and the game ends when one player has 0 dollars left, there after each player counts their money and the one with most money wins.

[3] Larman, 2005, side 219-220

Primary actors:
Player.

Secondary actors:
Christian (vores kunde)

Preconditions:
2-4 players.
Everyone starts at the field START.
Each player, depending on the amount of players, start with an amount of 2=20$, 3=18$, 4=16$.
Each player has the ability to buy fields that aren't special fields and hasn't been bought yet.
Player1, the youngest player, the one who starts.

Main flow:

1. Player rolls a die.
2. The die shows a value between 1-6.
3. The player moves the given dice value.
4. The player's account is updated depending of the field information.
    a. If the field is owned, pay tax to the owner
    b. If the field is open, the player can either buy or not.
    c. If the field have a message, do as the message says.
5. If player passes START field, the player receives 2$.
6. Next player's turn.
7. The game ends when one player have 0$ left and then the player with most money wins.

Postconditions:
The game ends when one player have 0$ left and then the player with most money wins.

Alternative flows:

1. If a player lands on "Train" field, the player gets an extra turn.
2. If a player lands on "Dolphin" field, the player pays 2$.
3. If a player lands on "Go to Prison" field, the player have to pay 1$ and move your car to the Prison field. Player does not receive 2 dollars if they pass START. Next time it is your turn, roll the dices and move from Prison field.
4. If a player lands on Prison field, nothing happens.
5. If a player lands on Free Parking, nothing happens.

Fully dressed:
Use Case ID 1: PlayGame:
Scope: Monopoly Junior
Level: User goal
Primary actors: Player1, Player2 optional Player3 and Player4
Secondary actor: Christian
Stakeholders and interests:

- Den sekundære aktør, Christian (vores kunde) ønsker et Monopolyprogram, der er tilpasset målgruppen 'børn'.
- Christian ønsker at programmet indeholder en spilleplade med forskellige felter, et raflebæger med én terning, 4 forskellig farvede brikker.
- Hver spiller har mulighed for at købe de felter, som ikke er specialfelter. Når et felt købes, indrammes det er samme farve, som spillerens brik har.
- Programmet skal kunne aflæse terningens værdi, og rykke den givne spille, samme antal felter, som terningens øjnene viser.
- Kunden ønsker at programmet skal kunne spilles af 2-4 spillere.
- Player1-Player4 (primary actor) ønsker at spille Monopolyspillet og vinde.
- Kunden ønsker, at den yngste af deltagerne starter.
- Player1-Player4 ønsker at kunne trykke på en knap, for at de to terninger bliver slået.
- Kunden ønsker at systemet kan aflæse terningens værdi og rykke en given spiller samme antal felt som er lig terningens øjnene.
- Player1-Player4 ønsker at kunne slå terningen af flere omgange.
- Kunden ønsker at spillet skal indeholde 25 felter, som indebærer felter, der kan købes af spillerne, felter som giver ekstra tur, og felter som gør at en spiller med sidde en tur over.
- Player1-Player4 ønsker at kunne trykke på en knap, for at købe et felt.
- Kunden ønsker, at systemet trækker et købsbeløb fra en given spiller.
- Kunden ønsker, at systemet kan aflæse, når en spiller lander på en modstanders købte felt.
- Player1-Player4 ønsker at blive betalt af sin modspiller, når/hvis han/hun lander på Player1-Player4's felt.
- Christian ønsker, at systemet kan aflæse, når en spiller lander på et 'Go to prison'-felt, og afregner en given spiller for 1$.
- Christian ønsker, at systemet kan aflæse, når en spiller lander på et 'Dolphin show'-felt, og afregner en given spiller for 2$.
- Christian ønsker, at systemet kan aflæse, når en spiller kommer over 'Start'-feltet og gives 2$.
- Player1-Player4 ønsker at få 2$ til sin pengebeholdning, når han/hun kommer over 'Start'-feltet.
- Christian ønsker at Monopoly spillet sluttes, når én af spillerne går bankerot. Det ønskes videre at den af de resterende spiller, som har flest penge tilbage, bliver udnævnt vinder.

Preconditions: Spillet er færdigt, når en vinder er fundet udfra, hvem der har flest penge tilbage.

Spillerne starter med henholdsvis 20$, 18$ eller 16$ alt efter, om de er 2, 3 eller 4 spillere ialt. Player1-Player4 starter spillet. Programmet skal bede om antal på spillerne, om deres navne, give dem mulighed for at vælge en farve, fortælle at det er den yngste spiller, der starter.

Success guarantee (or Postconditions): Spillet spilles indtil en vinder er fundet.

Main Success Scenario (or Basic Flows):

1. En fra Player1-Player4 trykker på 'Roll'-knappen og systemet respondere ved at vise værdien af terningens øjnene.
2. Systemet aflæser værdien af terningens øjnene og rykker den givne Player's brik samme antal felter.
3. En fra Player1-Player4 lander på et felt tilgængeligt for køb
4. En fra Player1-Player4 lander på et opkøbt felt.
5. En fra Player1-Player4 et 'Train'-felt.
6. En fra Player1-Player4 lander på et 'Go to prison'-felt.
7. En fra Player1-Player4 lander på et 'Dolphin show'-felt.
8. En fra Player1-Player4 lander på et 'Free parking'-felt.
9. Player1 kommer over 'Start'-feltet.

Player1-Player4 repetierer punkt 1-13 indtil:

10. Player1-Player4 går bankerot
11. Player1-Player4 udnævnes vinder.

Extensions (or Alternative Flows):

3a. For hver gang en spiller lander på et felt tilgængeligt for køb:

1. Spørger systemet, om spilleren vil købe feltet via en meddelelse og en købe-knap.
   1a. Køber spilleren feltet, trækkes beløbet på grunden fra den givne spillers pengebeholdning.
2. Hvis ikke spilleren køber feltet, går turen videre til næste spiller.

4a: Hver gang en spiller lander på et opkøbt felt:

1. Betales der "husleje" til ejeren af feltet.
   1a. Lander en spiller på et felt af én bestemt farve, som er opkøbt og hvor ejeren samtidig ejer en anden grund af samme farve, betales der dobbelt "husleje" til ejeren.

5a: Hver gang en spiller lander på et 'Train'-felt:

1. Udprintes en meddelelser om, at spilleren har fået en ekstra tur.
2. Den givne spiller slår igen.

6a: Hver gang en spiller lander på et 'Go to prison'-felt:

1. Skal den givne spiller betale banken 1$.
2. Rykkes den givne spiller til 'Prison'-feltet.

7a: For hver gang en spiller, lander på et 'Dolphin show'-felt:

1. Afgives en meddelelse om at banken trækker 2$ fra spillerens konto.
2. Trækkes der 2$ fra den givne spillers konto.

9a: Hver gang en spiller kommer over 'Start'-feltet:

1. Gives den givne spiller 2$ til sin pengebeholdning.

10a: Når en spiller går bankerot:

1. Udprintes en meddelelse, om at spillet at personen har tabt.

11a: Når en vinder skal findes mellem de resterende deltagere:

1. Aflæser programmet de sidste spilleres pengebeholdning.

1a. Hvis to eller flere af spillerne har lige mange penge tilbage, udregner programmet hvilken af de givne spillere, der har for mest værdi af købte felter.

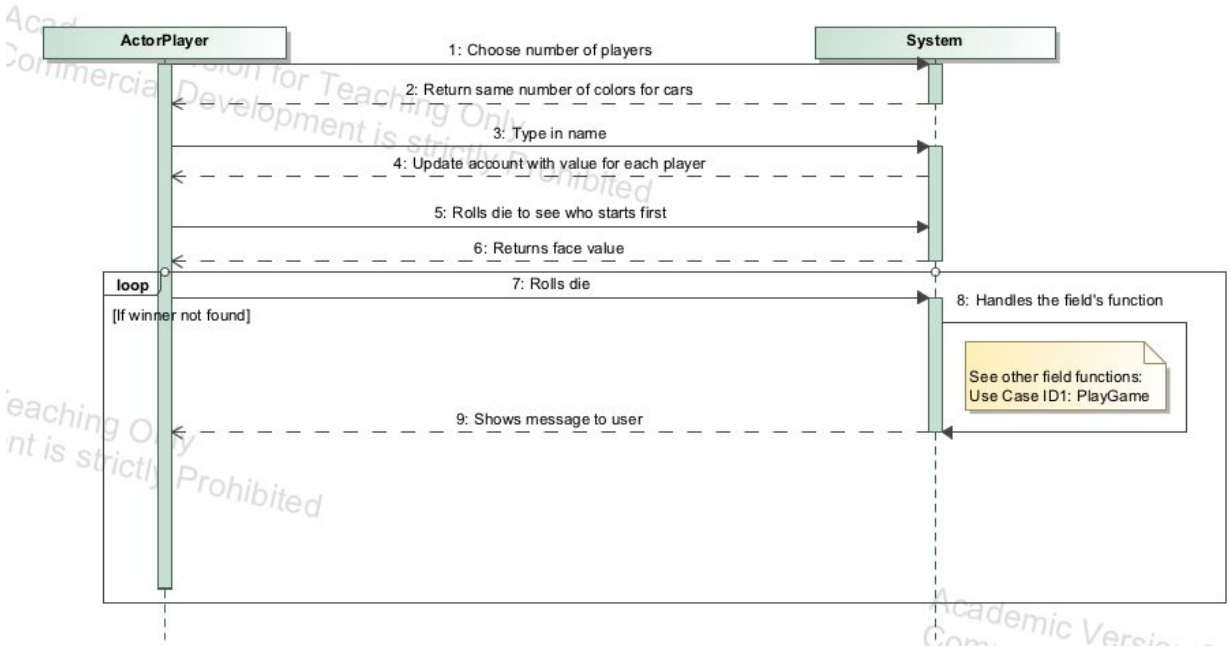
2. Når en vinder er fundet, udprinter systemet en meddelelse om hvilken brikfarve samt navn på spiller, der har vundet.

Special Requirements:

- Vi har valgt, at programmets responstid ikke må komme over 333 millisekunder.
- Programmet skal være visuelt tilgængelig for spillerne, samt at spillerne skal kunne interagere programmets funktioner via GUI (graphical user interface).
- Spillet skal være brugervenligt for børn og visuelt nemt at overskue.
- En pengebeholdning må ikke kunne blive negativ.
- Skal kunne spilles på DTU's databarer i Windows 10.

Technology and Data Variations List: Java, Eclipse 8, GUI & JUnit testing, Maven og Windows 10.

System sequence diagram:



System Sequence diagram lavet i MagicDraw

Arv:

Arv er den process hvor en klasse får egenskaber (metoder osv.) fra en anden klasse. Klasse der arver informationer er kaldt en subklasse mens klassen der nedarver kaldes en superklasse. Dvs at man definerer mere specificerede klasser ud fra eksisterende klasser. Det

er gjort til at forhindre kode duplikation og til at lave et bedre design der er nemmere at vedligeholde og teste[4].

Abstract:
Abstract klasser er klasser, der er "declared" `abstract`. Abstract klasser kan ikke instantieres, men de kan have subklasser. Abstract klasser kan også indeholde abstract metoder, men det er ikke nødvendigt, en klasse kan dog ikke indeholde abstract metoder uden at selv være en abstract klasse. De kan bruges hvis man skal dele kode mellem klasser der er (closely) relaterede til hinanden og skal bruge lignende metoder. [5]

Fieldklasserne ift. landOnField-metoden:
Polymorphism. Polymorfi er en del af GRASP principperne, der handler om at tildele ansvar, for at definere variationen af adfærd baseret på én klasse, superklassen, som videre er i korrelation til andre subklasser, for hvilke denne variation sker. Dvs. man bruger polymorfiske operationer til de typer, der laver forskellige ting. Vores landOnField metoden laver noget forskelligt på de forskellige felter, men de stammer alle fra den samme superklasse[6].

3. Design

GRASP (General Responsibility Assignment Software Patterns)

Creator:
Game, dicecup, player og gameboard klasserne er vores creatorere, dette skyldes at disse fire klasser opretter nye objekter. Fx. player opretter account fordi player objektet har en account som attribute.

Controller:

Game er vores controller grundet at hvis denne metode ikke var til stede, ville spillet ikke have mulighed for at køre. Controller skal være en der igangsætter programmet, med andre ord har den kørselsbetingelserne for vores program.

Høj samhørighed (High Cohesion):

Vores klasser har høj samhørighed da klasserne og dens objekter har et veldefineret ansvar.

Information Expert:

Game er vores informations ekspert, da klassen indeholder information om at kunne fuldføre mange af de andre klasser.

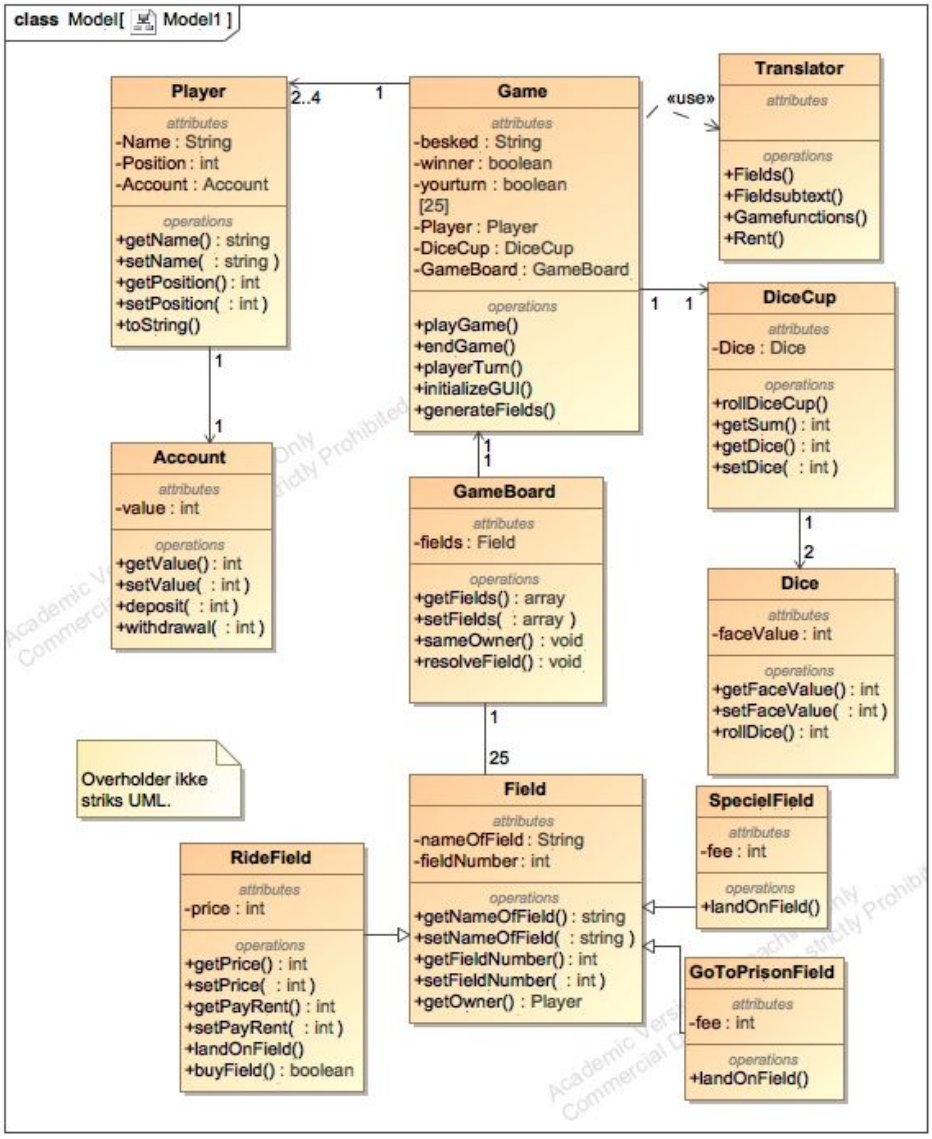
Lav kobling (Low Coupling):
Vores klasser har lav kobling da deres indflydelse på hinanden er ganske lav, account klassen er knyttet til player klassen, men dette skyldes at player har et account objektet som attribut.

[4] Ian Bridgwood, slides fra lektion 10 *Udviklingsmetoder til IT systemer.*
[5] Larman, 2005, side 392
[6] Larman, 2005, side 578

Ligeledes er dice knyttet til diceCup. Grunden til at det er relevant med lav kobling, skyldes at hvis der skulle være noget der går galt i en af klasserne, går hele programmet ikke ned.

Design Klasse Diagram:

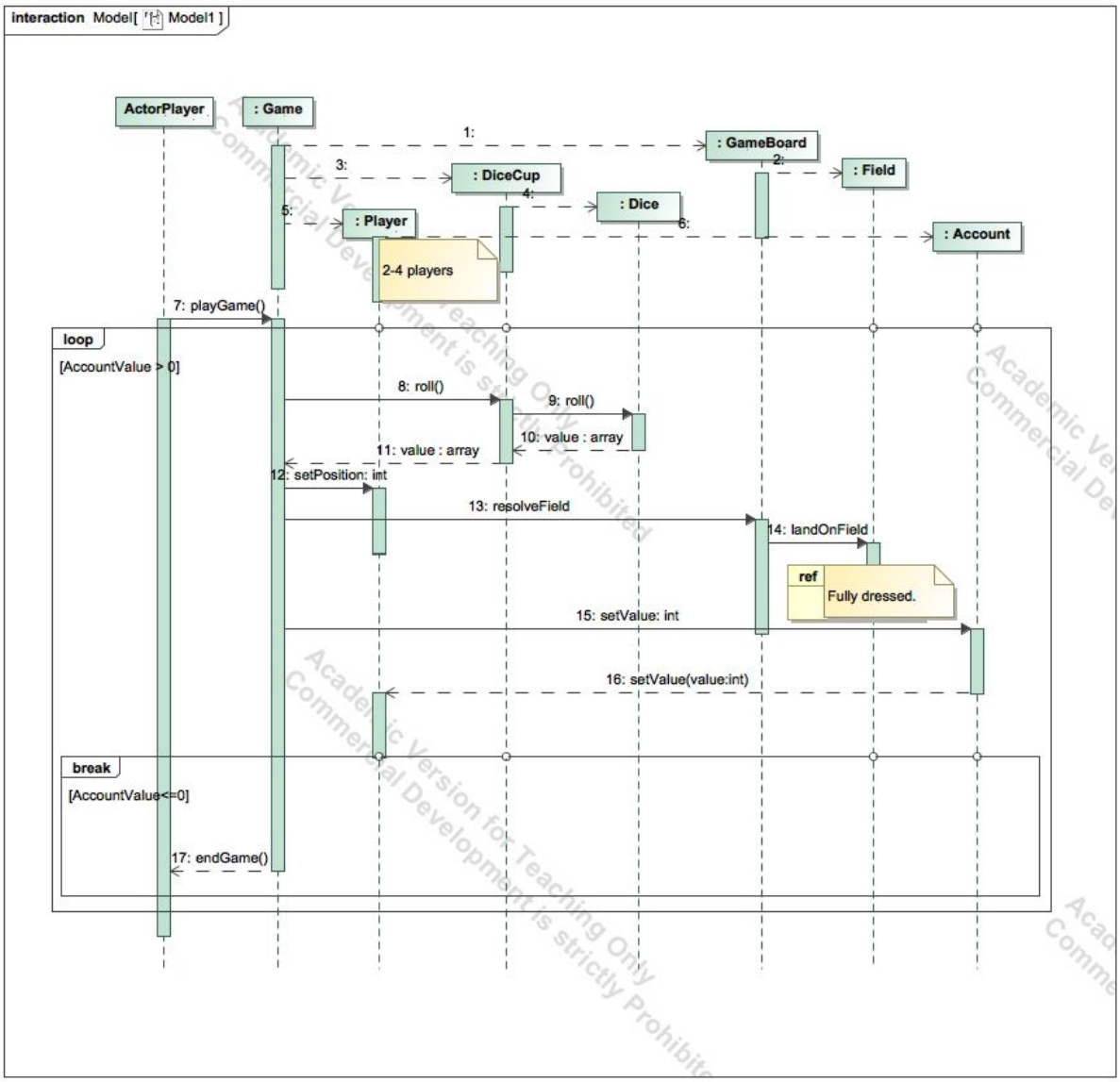
Vores Design Klasse diagram beskriver vores main metode spil og vores klasser. Som det kan ses på nedenstående figur, fortæller vores attributter hvad hvor klasser indeholder af variable, dernæst fortæller operationerne om hvilke metoder vi gør brug af i klasserne. Konto klassen kunne været i Spil klassen, men vi har knyttet Konto klassen til Spiller klassen, da vi synes der er sammenhæng i forhold til design og implementering.



Design class diagram lavet i MagicDraw

Design Sekvens Diagram:

For at visualisere vores programs in- og outputs har vi gjort brug af modellen Design Sequence Diagram. Der demonstreres her en specifik hændelse, der udføres af den eksterne aktør, som interagerer med softwaren for vores system, og videre det system, som aktøreren igangsætter.
Dét, som aktøren igangsætter er systemet hændelser (events), som leder til en system operation, som ligeledes kan håndterer disse hændelser (events)[7].
For at begrænse modellen, og de hændelser, der kan forekomme, har vi valgt at referere til vores fully dressed beskrivelse ift nogle af felters funktioner.
Vores model er derfor lavet for at give et overblik over vores system:



Design sequence diagram lavet i MagicDraw

[7] Larmann, 2005, side 173, 175, 176

4. Implementering

Da vi startede med programmeringen kiggede vi på hvor stor en del af vores kode fra sidste projekt der kunne genbruges. Vi endte med at genbruge Dice, Dicecup, Player, og Account klasserne. I sidste projekt havde disse klasser danske navne, så vi oversatte dem.

En af vores krav var at spillet skulle kunne spilles af 2 til 4 spillere. For at gøre dette muligt lavede vi et Array med spillere. startGame() metoden starter med at spørge brugeren hvor mange spillere de er, og laver derefter et Array med det givne antal pladser.

Vi oprettede en abstract Fieldklasse, og lavede nødvendige underklasser der håndterede hvad der skulle ske når man lander på hvert felt. I disse klasser er der høj kobling, som går imod vores anvendte GRASP-principper. Vi ville gerne rette dette, men vi har valgt at nedprioritere det for at løse mere pressende problemer. Alle disse Field Objekter bliver opbevaret i en GameBoard klasse. Vi har på grund af tidspres været nødsaget til at gøre vores subklasser til controllere. Hvis vi havde mere tid havde vi lavet controller-Klasser.

Vi valgte at dele hele spillet op i tre faser, hver især med deres egen metode; startGame(), playerTurn() og endGame(). Disse metoder håndterer oprettelse og afslutning af spillet samt Oprindeligt lavede vi vores Player-klasse om, så det var en subklasse af GUI_Player klassen, men vi fandt hurtigt ud af at dette var en dårlig ide, da disse to klasser har to forskellige ansvarsområder.

En af vores krav gik ud på at man skal betale dobbelt husleje hvis du felter af samme farve er ejet af samme spiller. For at få det til at virke har vi lavet en statisk metode der finder ejeren af begge felter som ligger ved siden af pågældende felt. Hvis en af disse felter ikke kan ejes, er der en metode i field-klassen der altid returnerer null.

Når en spillers konto rammer 0, og spillet skal afsluttes, starter endGame() metoden. Her oprettes først en ny spiller, “winner”. Herefter findes spilleren med flest penge på kontoen ved et forloop. Dette loop starter med at sammenligne den første spiller med winnerspilleren, som har 0 penge på kontoen. Hvis han har mere end 0 penge, bliver winners information overskrevet. Loopet tester kun om en spiller har flere penge end den forige spiller. Det vil sige at hvis to spillere har samme antal penge, vil den første spiller i vinder.

5. Test

Vi har lavet positive test ved hjælp af JUnit-test. Vores tests er lavet på baggrund af vores metoder i vores forskellige klasser, omhandlende basale funktioner for vores system. F.eks. om vores terning kan slå mellem 1-6, om vores addition og subtraktion fungere i account som det skal m.m.
Testen for Account, Player, RideField og Dice gik fint uden nogle fejl.
Vores RideField klasse var vi nødsaget til at opsætte GUI'en som parameter til metoderne, dette gjorde vi ved at indsætte det før testen kørte. Dette komplicerede dog testen, og gjorde den sværere at gennemføre, .
I vores test af RideField åbner vi GUI'en 5 gange, hvilket betyder at det tager længere tid at eksekvere testen.

Screenshot af tests er vedlagt som bilag.

6. Projektplanlægning

Som i de tidligere rapporter har vi valgt at bruge Unified Process til at dokumentere vores arbejdsproces.

Inception:
I denne fase konkluderede vi at programmet godt kunne laves, hvis vi tilføjede og sløjfede nogle krav. Vi lavede her vores use-case diagram 'PlayGame' og fandt de relevante aktører.

Elaboration:
Vi har taget udgangspunkt i Monopoly Junior spillereglerne og stillet dem op som vores krav efter FURPS metoden. Denne gang havde kunden bedt os om at lave et Use Case diagram, et domænemodel, et system sekvens diagram, et design sekvens diagram og et design klasse diagram. Derudover skulle vi have en use case beskrivelse (fully dressed) og en beskrivelse over de GRASP principper vi har overholdt. Desuden var der en del dokumentation kunden ville have i rapporten som en definition af Arv og Abstract og dokumentation af tests.
I løbet af denne fase og i forbindelse med kodningen, valgte vi at sløjfe et par af de krav, som vi egentlig havde nedskrevet i FURPS i første omgang.
Da vi ikke kunne få vores GUI til at sætte andet end grønne huse på felterne, valgte vi i stedet at indramme feltet med en given spillers farve, når han/hun købte det.
Derudover så vi det unødvendigt at lave en begrænsning for, hvor mange felter, der kunne købes, da det næsten ville være usandsynligt, at en spiller ville nå at købe 12 felter (som var den oprindelig regel), siden spillepladen består af så få felter.

Construction:
Ved er klar over, at der er høj kobling i vores program, da vi har en metode i alle Field klasserne, som bruger GUI'en. Vi havde rettet dette, hvis vi havde opdaget det tidligere, og hvis der havde været tid til det, men det var der på dette tidspunkt ikke.
Ift testing lavede vi via JUnit en positiv test, se under punkt 5 for yderligere uddybning.

Transition:
Da vi havde lavet størstedelen af strukturen for programmet, fik vi en tilfældig person til at teste vores spil. Da denne person ikke forstod, hvad de specifikke regler var, valgte vi at indsætte flere meddelelser i spilles felts funktioner. Hertil valgte vi også at lave en medfølgende manual til spillet, som vi også mener ville være nødvendig for typiske børnespil i den virkelige verden.

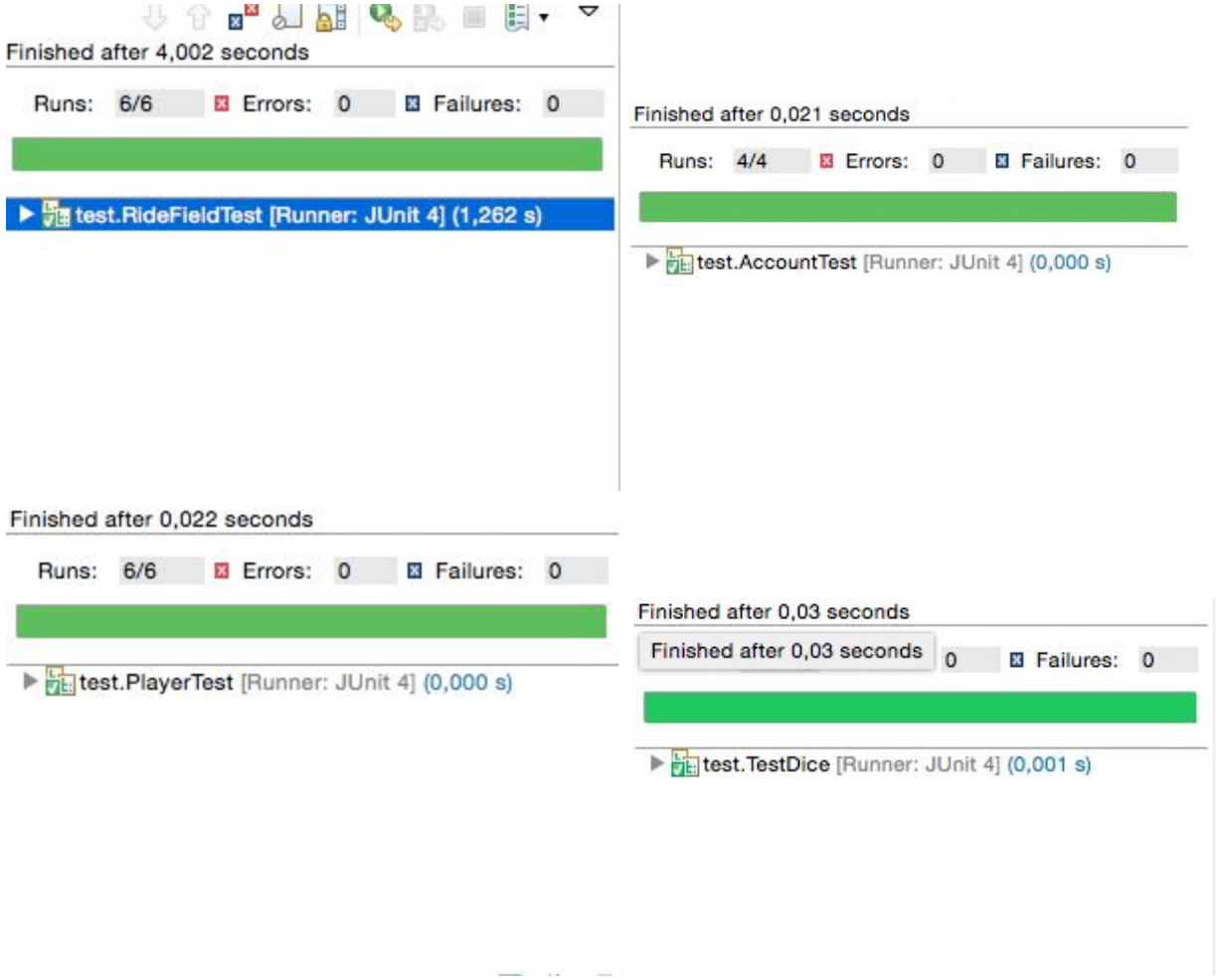
7. Konklusion

Formålet med opgaven har været at lave et program baseret på brætspillet Monopoly Junior.
Vi har et færdigt produkt som opfylder de krav der tidligere i rapporten er blevet beskrevet, dog med nogle ændringer fra de originale krav, som vi syntes ville forbedre programmet. Vi har gjort brug af Unified Process til at beskrive arbejdsprocessen, ændringerne af kravene blev beskrevet i elaboration fasen.

8. Litteratur og kildeliste

- ***Larman, Craig. (2005)***: 'Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and Iterative Development'. (3. udgave). Pearson Education Inc.

Bilag


Finished after 4,002 seconds
Runs: 6/6
Errors: 0
Failures: 0
test.RideFieldTest [Runner: JUnit 4] (1,262 s)
Finished after 0,021 seconds
Runs: 4/4
Errors: 0
Failures: 0
test.AccountTest [Runner: JUnit 4] (0,000 s)
Finished after 0,022 seconds
Runs: 6/6
Errors: 0
Failures: 0
test.PlayerTest [Runner: JUnit 4] (0,000 s)
Finished after 0,03 seconds
Finished after 0,03 seconds
0
Failures: 0
test.TestDice [Runner: JUnit 4] (0,001 s)