CDIO Final - Matadorspil



Andreas Mads Nøhr	Athusan Kugathasan	Patrick Lopdrup Hansen
s185093	s185098	s185092


Lars Löfvall Redemann	Martin Rune Rylund
s185094	s185107

Link til Repository: https://github.com/mrylund/19_final
Antal sider i rapport: 38

Indholdsfortegnelse

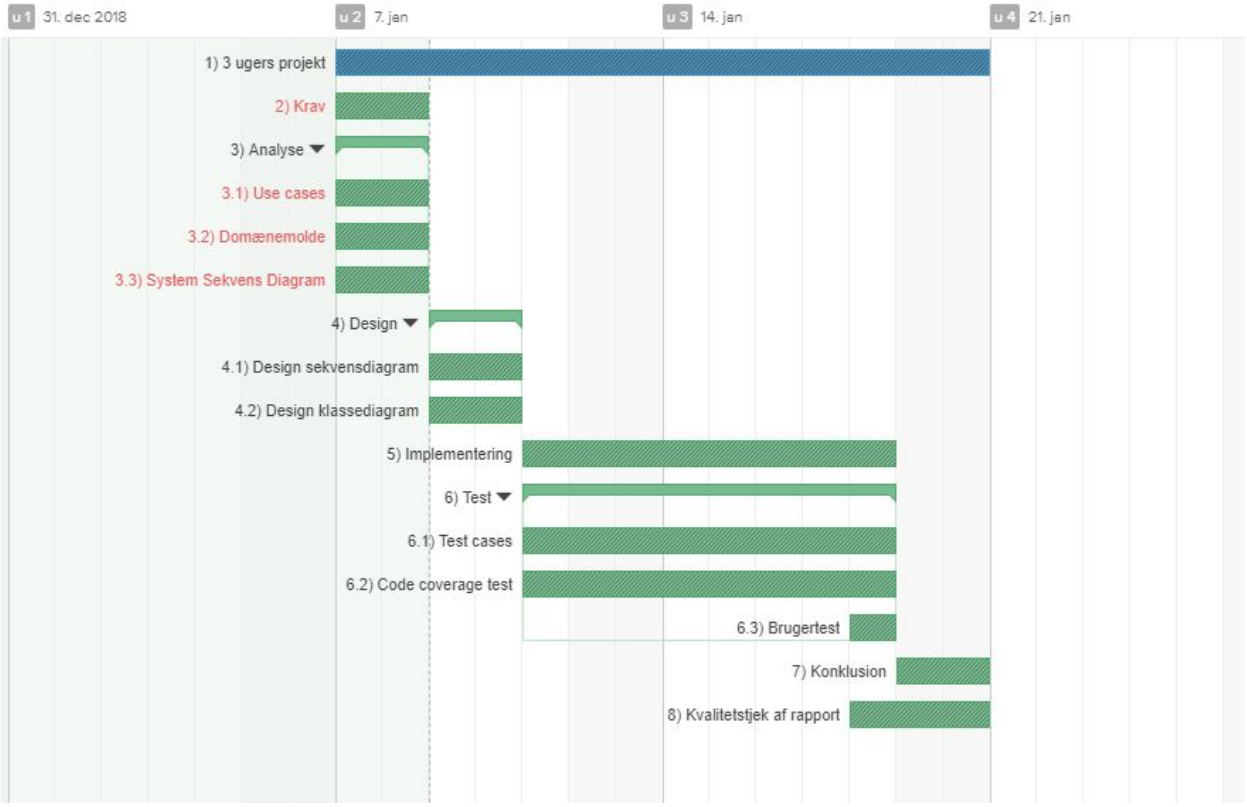
Timeregnskab

Dette regnskab viser det samlede antal af timer hver af gruppen medlemmer har arbejdet gennem 3-ugers perioden.

Dato/navn	Lars	Patrick	Athusan	Martin	Andreas
Mandag 07/01	4	4	4	4	4
Tirsdag 08/01	4	4	Syg	3	4
Onsdag 09/01	3	3	3.5	4	4
Torsdag 10/01	3	3.5	3.5 (forsinket)	3	(arbejdet hjemme)
Fredag 11/01	3(syg)	4	4	5	4
Mandag 14/01	3(syg)	Syg	4	3	syg
Tirsdag 15/01	4	4	4	4	4
Onsdag 16/01	3	5	5	5	3
Torsdag 17/01	(Arbejdet hjemme)	4	4	4	5
Fredag 18/01	(Arbejdet hjemme)	hjemme	Hjemmearbejde	Hjemme	(arbejdet hjemme)
Hjemme-arbejde	24	24,5	28	28	30
Samlet antal timer	51	56	60	63	58

Gantt-Tidsplan

Vi har i starten af forløbet udarbejdet en tidsplan for de to uger vi har til at skrive en rapport, samt kode et velfungerende Matadorspil.
I tidsplanen har vi sat små "milestones" for visse ting der skal være færdige til bestemte tidspunkter.



Indledning

Vi har fået en sidste opgave om at udvikle en version af det fulde matadorspil, som er en større version af vores sidste opgave, junior matador spillet. Vi har fået en bemærkning om at kunden hellere vil have et velfungerende program der ikke kan så meget end et program der kan en masse ting som ikke fungere.

Kundens vision

Kunden ønsker et fuldt ud udviklet matadorspil, med udgangspunkt i det normale matadorspil fra BRIO. Vi vil ved hjælp af en MoSCoW selv vurdere hvad der er vigtigt, for at kunne kalde det et matador spil og hvad der er realistisk at få implementeret på de 10 dage (foruden weekenden) vi har at programmere i.

Analyse af ovenstående problemstilling

Prioriteringsliste/kravliste

Vi bruger MoSCoW til prioritering af kravene.

Must have:

- **K1:** Slå med 2 terninger
- **K2:** Spillerne skal kunne bevæge sig på brættet.
- **K3:** Skal kunne spilles på DTU's databarer.
- **K4:** Lander man på et felt vha. terningekastet eller lykkekort og feltet ikke ejes af andre, kan man selv købe det.
- **K5:** Spillere starter på "Start" feltet .
- **K6:** Koden skal skrives i java.

Should have:

- **K7:** Alle spillere starter med 30.000 kr
- **K8:** Alle ens grunde skal ejes før der kan købes huse.
- **K9:** Inden man opfører et hotel, skal der være 4 huse på 4 grunde med samme farve.
- **K10:** Der må kun være ét hotel på hver grund og de 4 huse bliver indleveret til banken
- **K11:** Passerer man start, får man 4000 kr.
- **K12:** Lander man på "prøv løkken", trækker man det øverste kort fra lykkekorts bunken

- **K13:** Hvis man er i fængsel og slår 2 ens, kommer man ud af fængslet og rykker antal øjne man slog
- **K14:** Vi skal genbruge så mange dele af kode fra tidligere CDIO afleveringer som muligt.
- **K15:** Koden skal være overskuelig så andre kan genbruge den

Could have:

- **K16:** Der skal være ét hus på hver grund, før der kan bygges endnu et hus.
- **K17:** Huse og hoteller bliver solgt til halv pris til banken.
- **K18:** Der er et specifikt antal huse og hoteller i spillet. 30 huse og 10 hoteller.
- **K19:** Man har 3 ture til at slå to ens med terningerne; dette har man tre omgange til. Hvis man ikke slår to ens nogle af de 9 gange, betales 1000kr og man kommer ud af fængslet.
- **K20:** Hvis man vil handle indbyrdes skal alle huse sælges før man handle grundene.
- **K21:** Skylder en person mere end han ejer overgiver personen alt han/hun ejer til personen han/hun skylder penge og spilleren er ude af spillet.(skylder personen til banken sættes grundene til auktion mellem de andre spillere)
- **K22:** Det er muligt at pantsætte sine grunde for halvdelen af opkøbsprisen.
- **K23:** Imens en grund er pantsat er det ikke muligt at opkræve leje for grunden.
- **K24:** Ønsker man at tilbagebetale sit lån betaler man en rente på 10% (Har man en grund til en opkøbspris på 2000 kr får man en pantsætning på 1000kr når man tilbagebetaler banken. Man betaler altså rente af pantsætningen. 1000 + 10% af 1000 kr, så 1100 kr)
- **K25:** Det ikke muligt at pantsætte grunde med huse eller hoteller, disse skal sælges til banken inden grunden pantsættes.
- **K26:** Hvis en spiller lander på et felt og ikke køber det, sættes det på auktion.
- **K27:** Er man i fængsel, har man stadig mulighed for at byde ind på auktion og opkræve leje af grunde.
- **K28:** Spiller man under 6 spillere skal man kunne spille med “AI’s” eller “bots”

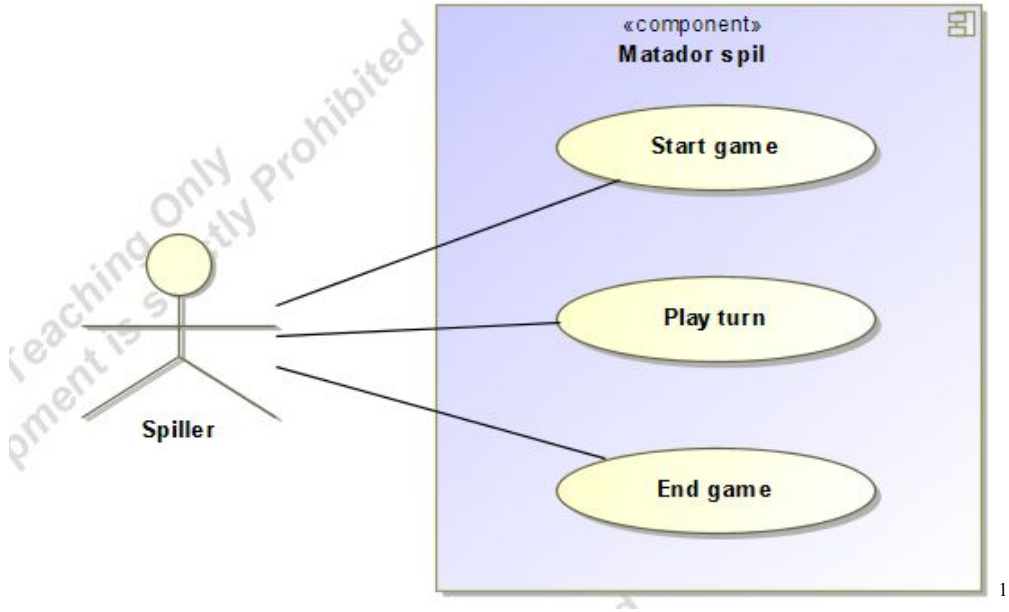
Won’t have:

- **K29:** Highscore
- **K30:** Man skal selv huske at indkræve leje

Use cases

- Start spil
- Spil tur
- Afslut spil

Use Case Diagram



[1]

Ovenstående billede er en visualisering af de use-cases en bruger kommer ud får ved brug af systemet.
En bruger vælger først at starte spillet, hvorefter der indtastes krævede oplysninger.
Derefter vil brugerens tur starte, og spillet vil hermed være sat i gang.
Til sidst, når en spiller går fallit, så findes der en vinder.

[1] Inspireret af vores eget diagram fra CDIO3

Use case beskrivelser

Briefly dressed use-case beskrivelse - Start game

Antallet af spillere indtastes og hver spiller indtaste sit navn

Fully-dressed use case beskrivelse - Play turn

Use Case Section	Play turn
Scope	Vores matadorspil
Level	Spilleren vil spille hans tur
Primary Actor	Spillerne
Stakeholders and Interests	IOOuterActive(Vores virksomhed, som laver spillet til vores kunde)
Precondition	Spillet er startet og alle spillerne er startet med 30000 kr.
Main success scenario	1. Spiller A kaster med to terninger 2. Spiller A rykker sin brik efter summen af terningerne 3. Spiller A lander på et ejendomsfelt 4. Spiller A kan vælge at købe ejendommen eller lade vær 5. Spiller As saldo ændres afhængig af hvad spilleren vælger 6. Det er nu næste spillers tur - Start fra 1 indtil det igen er Spiller As tur 7. Gå til trin 2 a. Hvis Spiller A passerer start feltet skal der tilføjes 4000 kr til Spiller As saldo 8. Gentag indtil alle udover én spiller er gået fallit
Alternate flows	A) En spiller kaster med terningerne a) Spilleren lander på et ejendomsfelt i) Spilleren køber feltet (1) Feltet pris bliver trukket fra spillerens saldo (2) Spilleren ejer nu feltet (3) Næste spillers tur ii) Spilleren køber ikke feltet (1) Feltet bliver sat på auktion (2) Næste spillers tur iii) Feltet er ejet af en anden spiller (1) Spilleren betaler leje til feltets ejer (2) Spillerens saldo ændres (3) Ejerens saldo ændres

	(4) Næste spillers tur iv) Spilleren ejer i forvejen dette felt (1) Spilleren afslutter sin tur på feltet (2) Næste spillers tur b) Spilleren lander på fængslet i) Spilleren er på besøg (1) Næste spillers tur ii) Spilleren ryger i fængsel (1) Spilleren har ikke slået to ens på 3 ture og betaler 1000 kr (a) Spilleren kommer ud af fængslet (b) Næste spillers tur (2) Spilleren bruger et frikort (a) Spilleren kommer gratis ud af fængslet (b) Næste spillers tur (3) Spilleren slår to ens (a) Spilleren kommer ud (b) Næste spillers tur c) Spilleren lander på chance feltet i) Spilleren trækker et tilfældigt kort (1) Spilleren får et pengebeløb (a) Spillerens saldo ændres (b) Næste spillers tur (2) Spilleren skal betale et pengebeløb (a) Spillerens saldo ændres (b) Næste spillers tur (3) Spilleren skal rykke til et felt (a) Ryk i fængsel (b) Lander på et ejendomskort - Tilbage til A (a) (c) Næste spillers tur (4) Spilleren trækker et frikort d) Spilleren lander på start feltet i) Spilleren får 4000 kr tilføjet til saldoen ii) Næste spillers tur e) Spilleren lander på parkering i) Intet sker f) Spilleren slår to ens i) Spilleren får en ekstra tur når hans tur slutter ii) Når ekstraturen er slut, er det næste spillers tur
Special Requirements	Spillet er startet og en spillers tur er lige startet
Frequency of Occurrence	Hver gang en spiller skifter turen videre til en anden spiller.

Briefly dressed use case beskrivelse - End game

Når alle, på nær én spiller, går fallit, har denne spiller vundet.
Vinderen skrives på skærmen.

Navneords-analyse på fully dressed use-case beskrivelse

1. Spiller
2. spillers tur
3. kaster terninger
4. terninger
5. sum af terninger
6. køber / køber ikke felt
7. feltet
8. næste spillers tur
9. sat til auktion
10. feltet er ejet af en anden spiller
11. betaler leje
12. saldo
13. afslutter tur
14. ryger i fængsel
15. betaler 1000 kr for at komme ud af fængslet
16. frikort
17. slår to ens
18. ekstra tur
19. chance felt
20. trækker tilfældigt lykkekort
21. går over start feltet
22. parkeringsfelt

Kandidater til Klasser

1. Spiller
2. Terninger

Kandidater til metoder

1. spillers tur()
2. kaster terninger
3. køber / køber ikke felt
4. næste spillers tur
5. sat til auktion
6. feltet ejes af en anden spiller

7. betaler leje
8. ryger i fængsel
9. slår to ens
10. ekstra tur
11. trækker lykkekort
12. går over start felt
13. terninge sum

Kandidater til attributter

1. felt
2. leje
3. beløb for at komme ud af fængsel
4. chance felt
5. start felt
6. parkeringsfelt
7. hus
8. hotel

PI (probability/impact) matrix

Liste over risici:

Vi vælger at kigge på vores "must have" krav og lave en liste over disse.

1. Spillet virker ikke på computerne i databaren.
   Improbable (1) og significant (4) = 5 (monitor)
2. Hvis vi ikke bliver færdige til tiden.
   Improbable (1) og catastophic (5) = 6 (monitor)
3. Hvis gruppen ikke overholder aftaler.
   Remote (2) og moderate (3) = 5 (monitor)
4. Vi får ikke implementeret alle "must have" krav, som vi har udarbejdet.
   Improbable (1) og significant (4) = 5 (monitor)

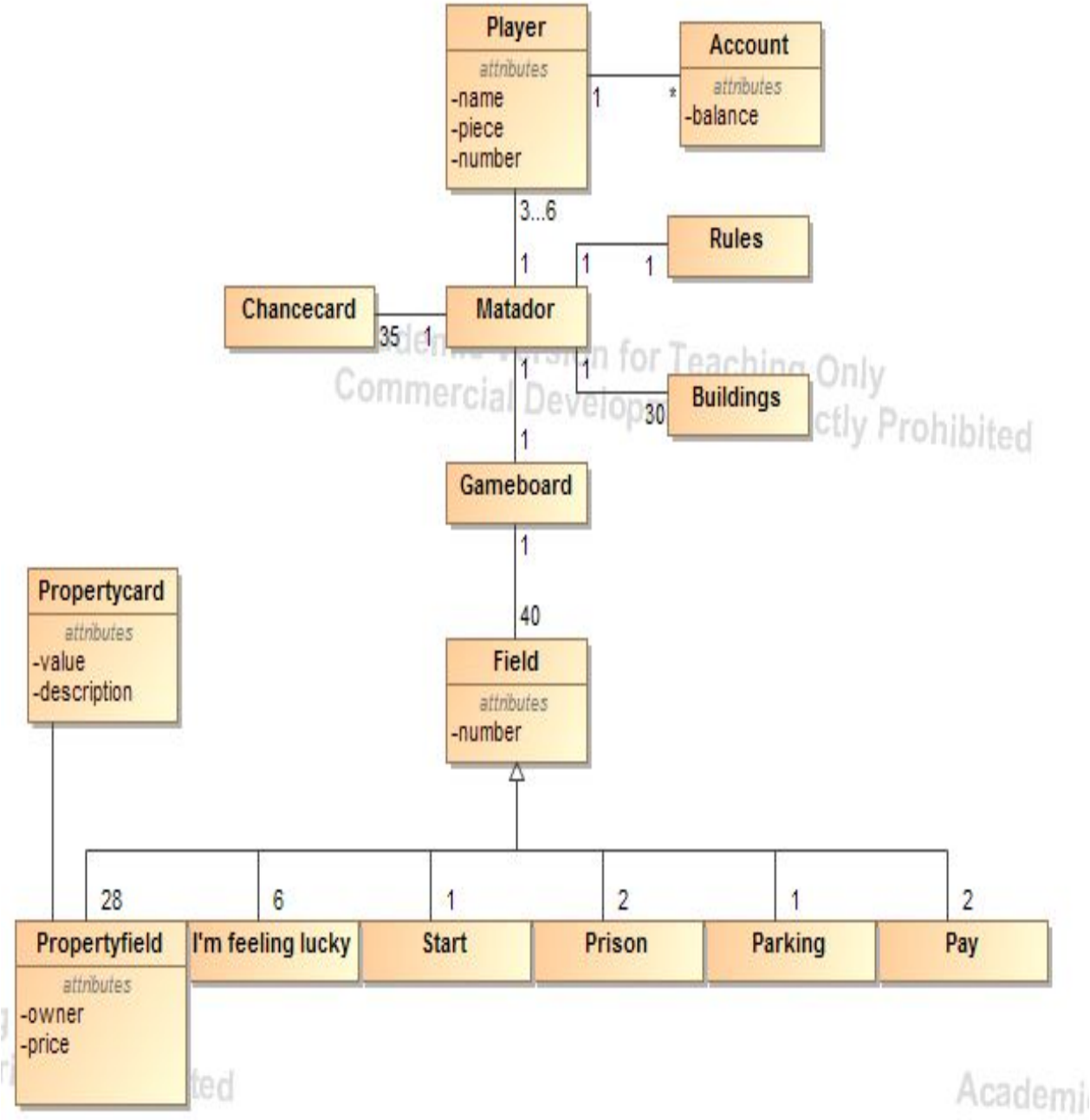
Catastrophic	5	10	15	20	25
Significant	4	8	12	16	20
Moderate	3	6	9	12	15
Low	2	4	6	8	10
Negligible	1	2	3	4	5
	Improbable	Remote	Occasional	Probable	Frequent

Catastrophic		Stop
Significat		Urgent action
Moderate		Action
Low		Monitor
Negligible		No action

Vi har ikke nogle specielt svære krav. Det er blot noget, vi skal holde øje med. Dog er risikoen for at de forskellige risici sker meget lav.

Domænemodel

Denne model er en visuel repræsentation af virkeligheden. Altså et fysisk matadorspil. Modellen viser hhv. hvilke klasser der interagerer med hinanden og deres multipliciteter. Nogle af klasserne indeholder attributter. Eksempelvis indeholder Player klassen attributterne; name, piece og number. Systemet bruger navnet og brikken til at identificere hver spiller og det giver et bedre overblik når man spiller spillet. Vi beskriver metoder i klasserne senere i Design Klasse Diagrammerne.

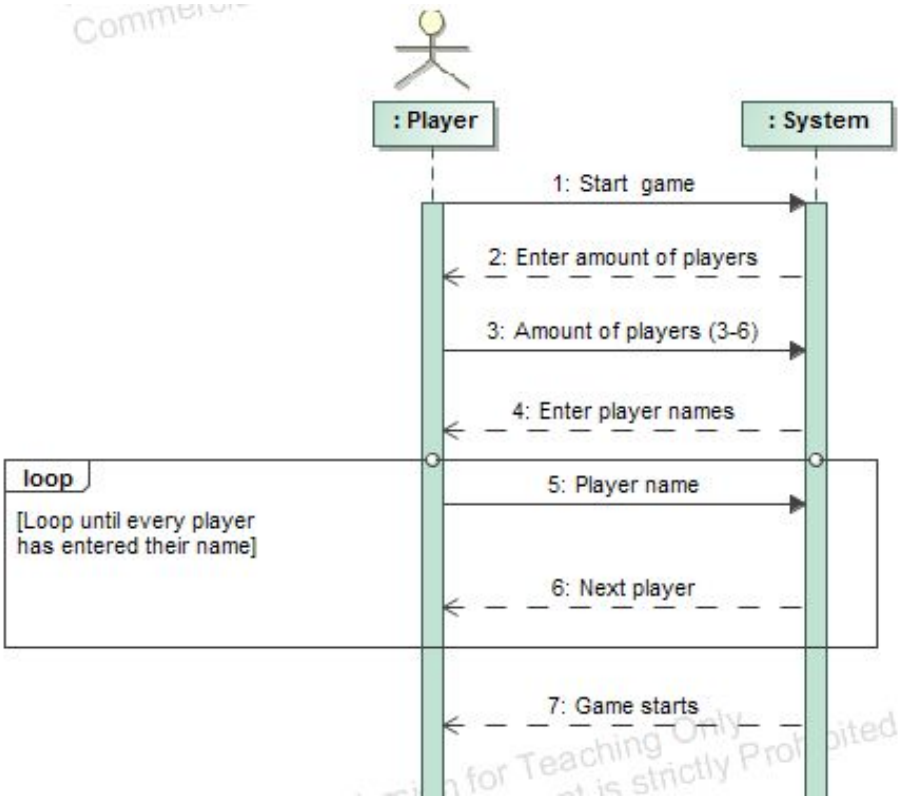


System Sekvens Diagrammer

Use case: Start game

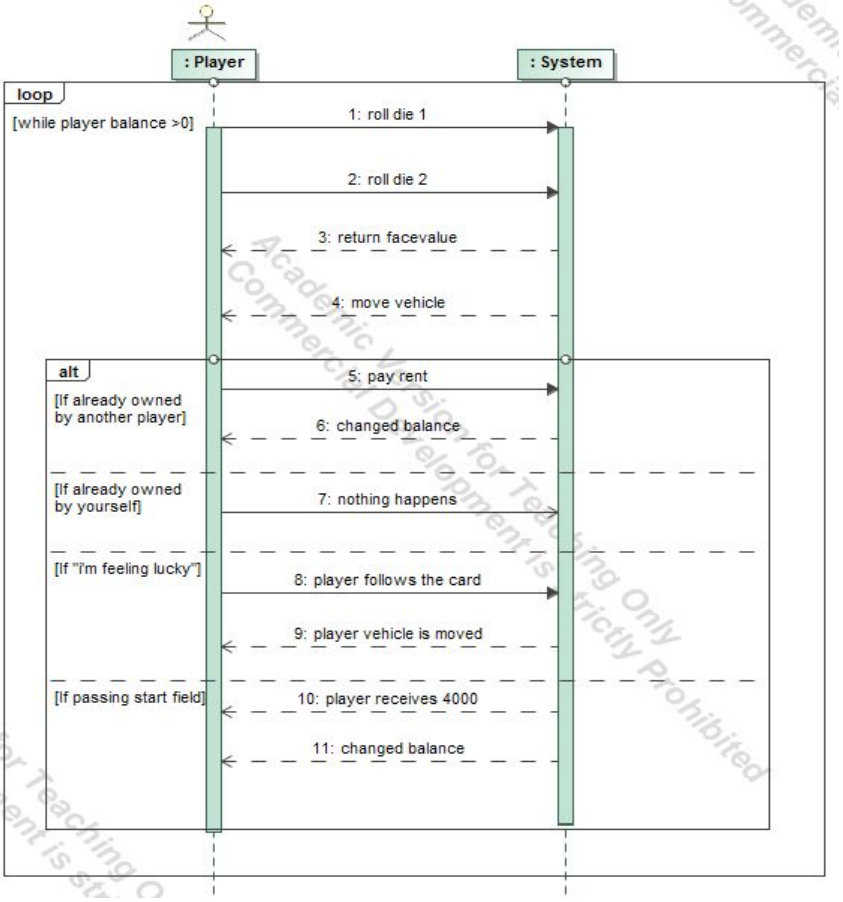
Dette Sekvens diagram viser interaktionen mellem spilleren og systemet, ved opstart af matadorspillet.

Ved start af spillet får spilleren mulighed for, at indtaste antallet af spillere. Hver spiller indtaster så sit navn og sekvensen gentager sig selv, indtil alle navne er indtastet. Når systemet kender til alle spillernes navne, skal hver spiller vælge en brik. Denne sekvens gentager også sig selv, indtil alle spillere har valgt en brik. Herefter starter spillet og hver spiller starter med 30.000 kr. i deres saldo.



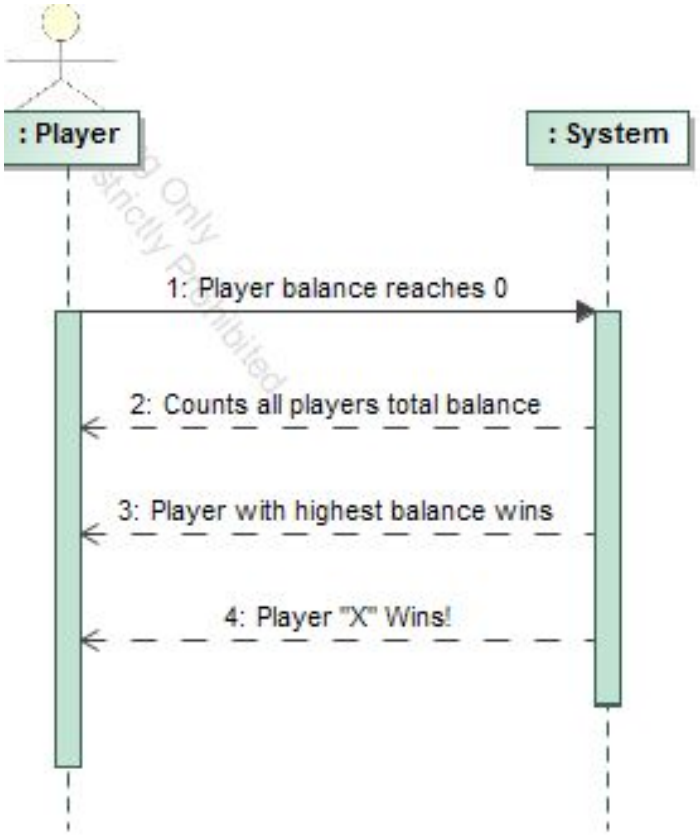
Use case: Play turn

Viser interaktionen mellem spilleren og systemet når en spiller skal spille en runde.
Når spilleren starter sin tur, kaster han med de to terninger i raflebægret. Herefter vil systemet returnere antallet af øjne på terningerne og derefter rykke spillerens brik. Herfra kan der opstå mange scenarier, alt afhængig af, hvor spilleren lander. Herefter opstår en ny sekvens mellem spiller og system. Når sekvensen er færdig, går turen videre til næste spiller og den samme sekvens gentager sig igen. Sekvens går rundt i et loop indtil en spillers saldo går i nul.



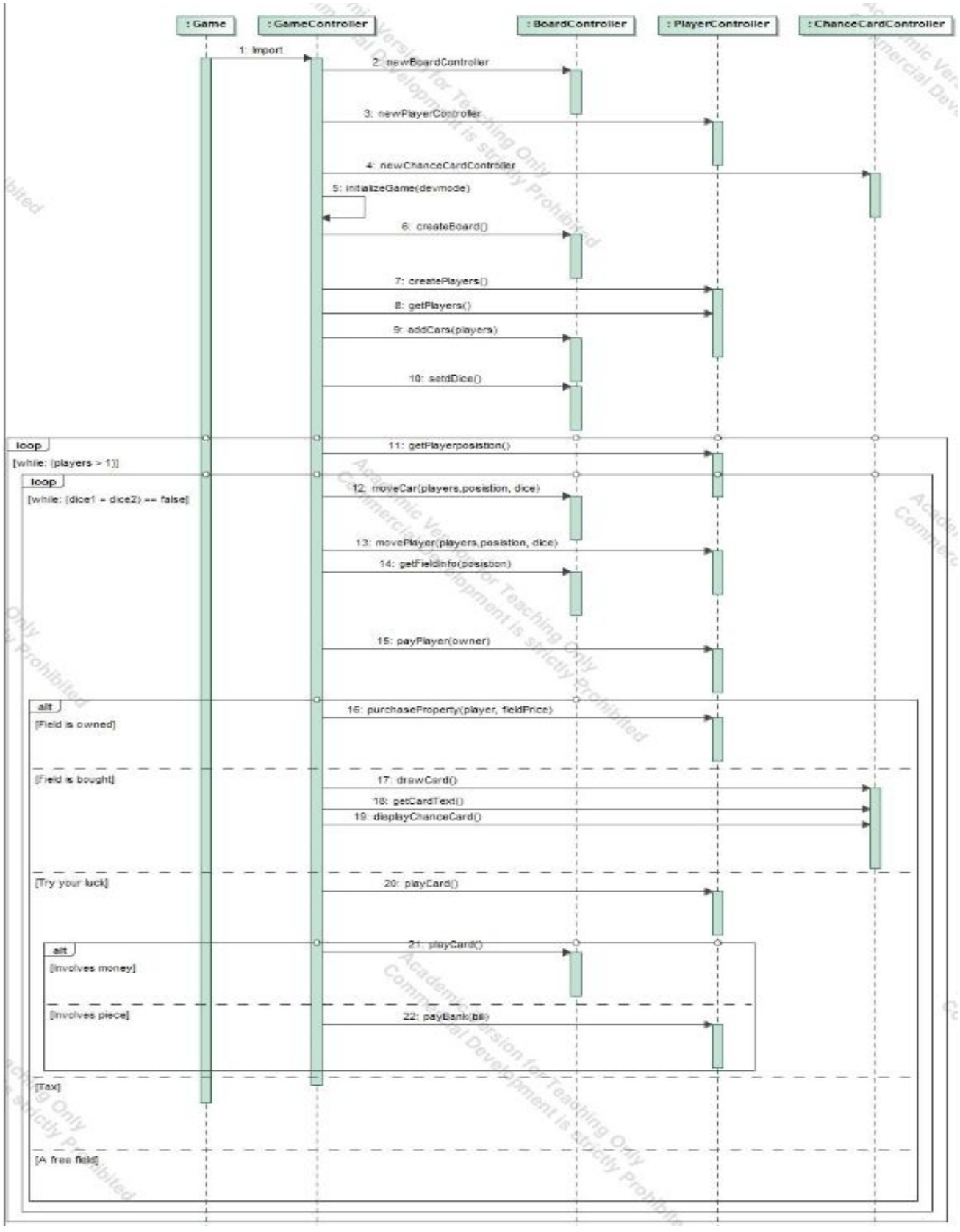
Use Case: End game

Viser sekvenses mellem spillerne og systemet når spillet afsluttes. Spillet afsluttes når en spiller går fallit. Herefter skal systemet tælle alle resterende spilleres saldo sammen. Efter beregningen har spilleren med den største saldo vundet og vinderen bliver vist frem på brugergrænsefladen.



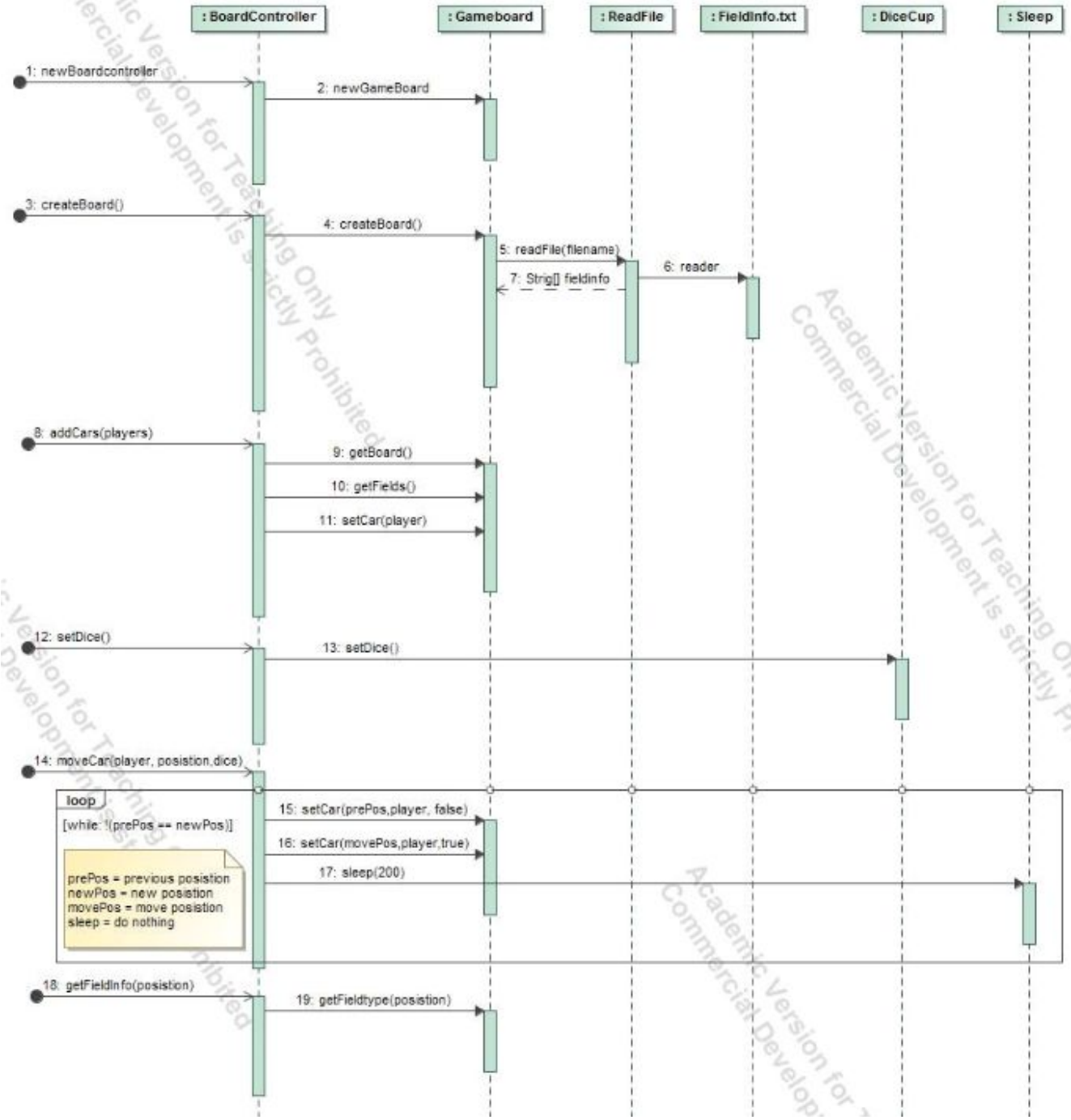
Design af vores kode-implementering

Design Sekvensdiagrammer - **(Controller + DiceCup sekvensdiagram)**



På ovenstående sekvensdiagram, kan man se interaktionen mellem de forskellige controllere. Heriblandt er Gamecontrolleren som styre spillet og har kendskab til de andre controllere. Derudover er Dicecup klassen også vist i diagrammet, på trods af det ikke er en controller. Vi har valgt at vise dette sammen med controllerene da det gør diagrammet nemmere at forstå og læse.

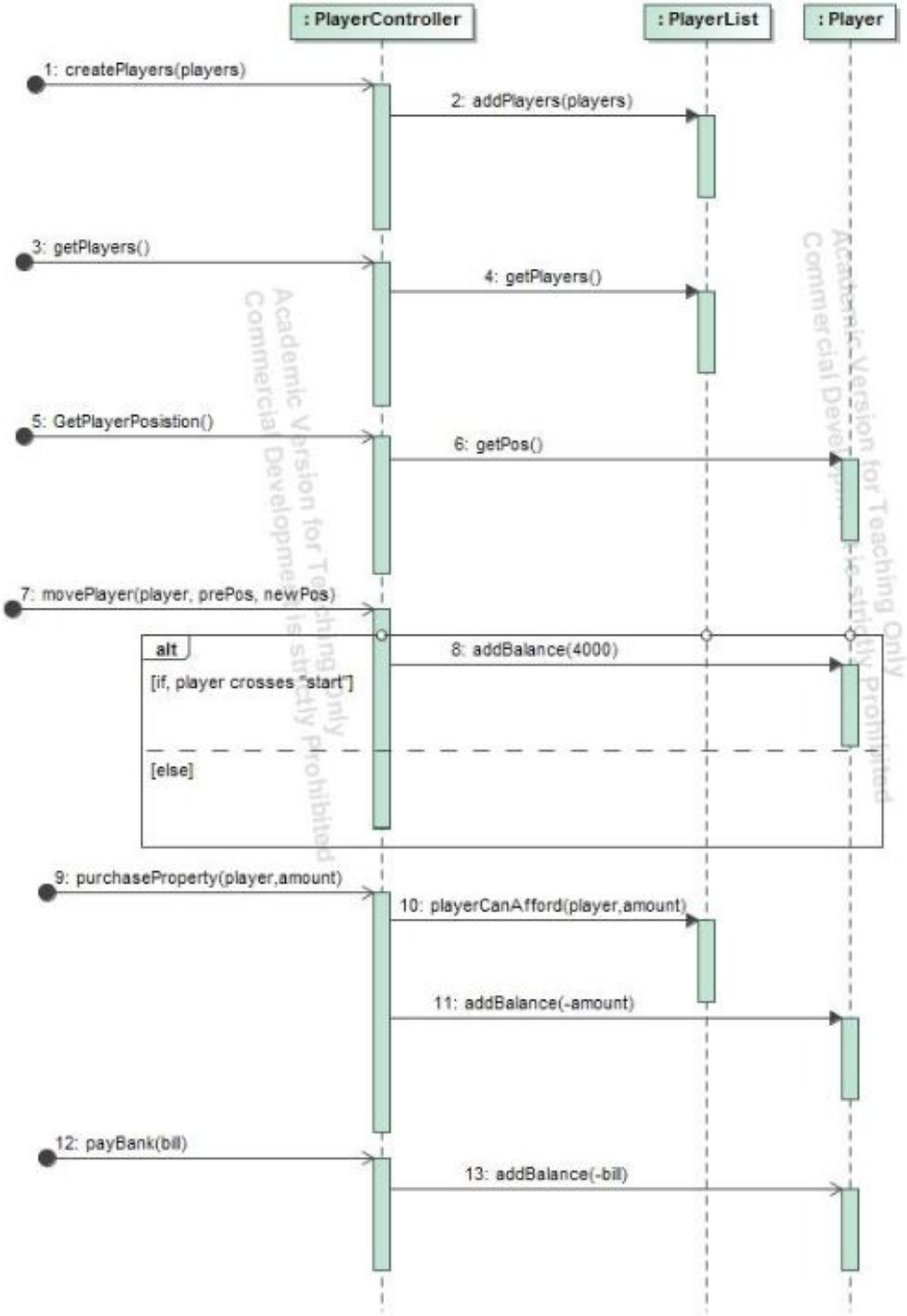
BoardController sekvensdiagram



På ovenstående diagram, er Boardcontrolleren illustreret. Boardcontrolleren er controlleren vi har lavet til at kontrollere det der sker på brættet. Metoden createBoard() er lidt speciel da det

er her vi får programmet til at læse alt informationen omkring felterne på brættet. Det sker ved vi har skrevet alle felterne ind i et tekstdokument og givet hver felt deres egen linje. På den måde kan vi nemt ændre på priser og navne. Dette dokument er skrevet som :FieldInfo.txt i diagrammet.

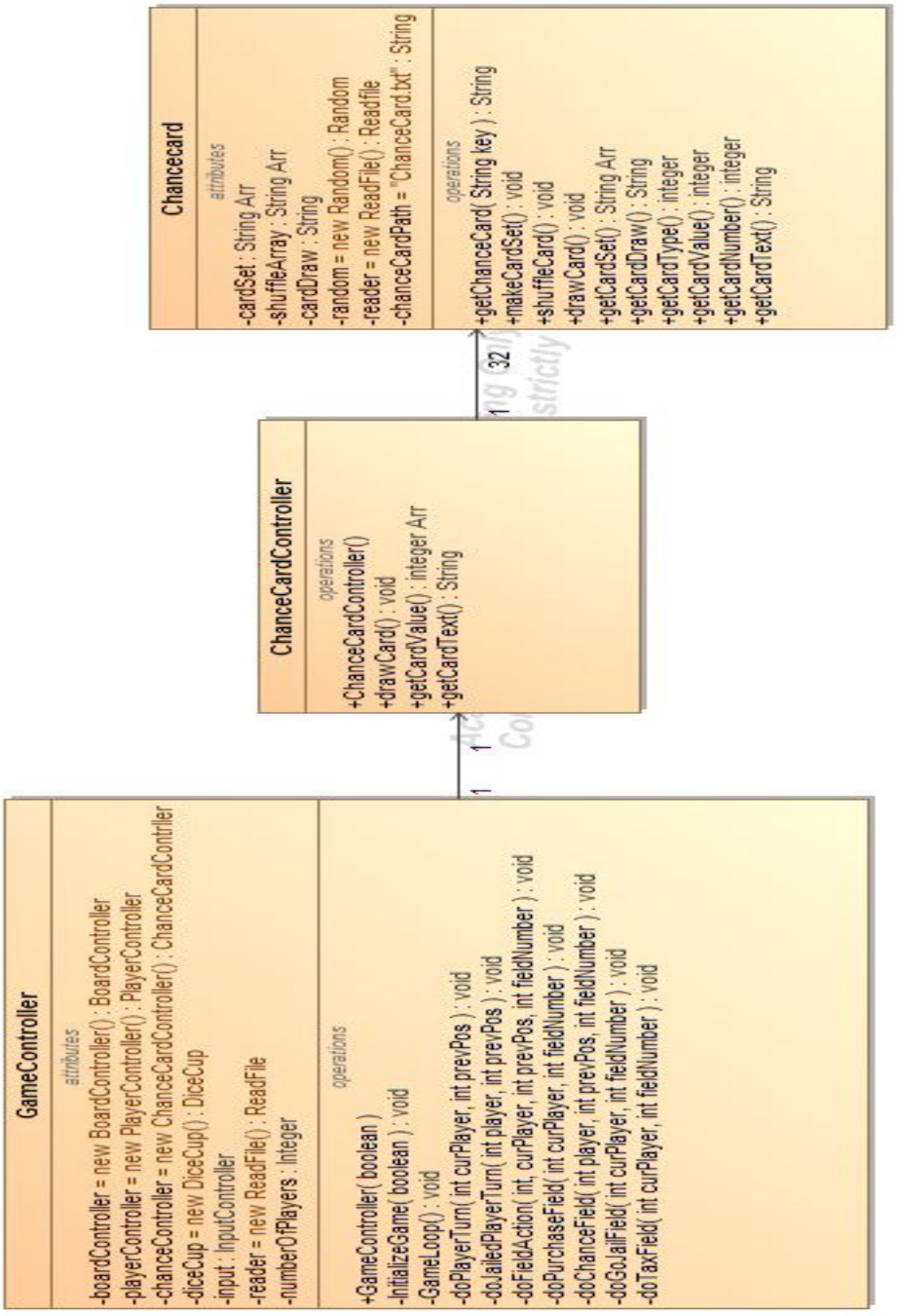
PlayerController sekvensdiagram

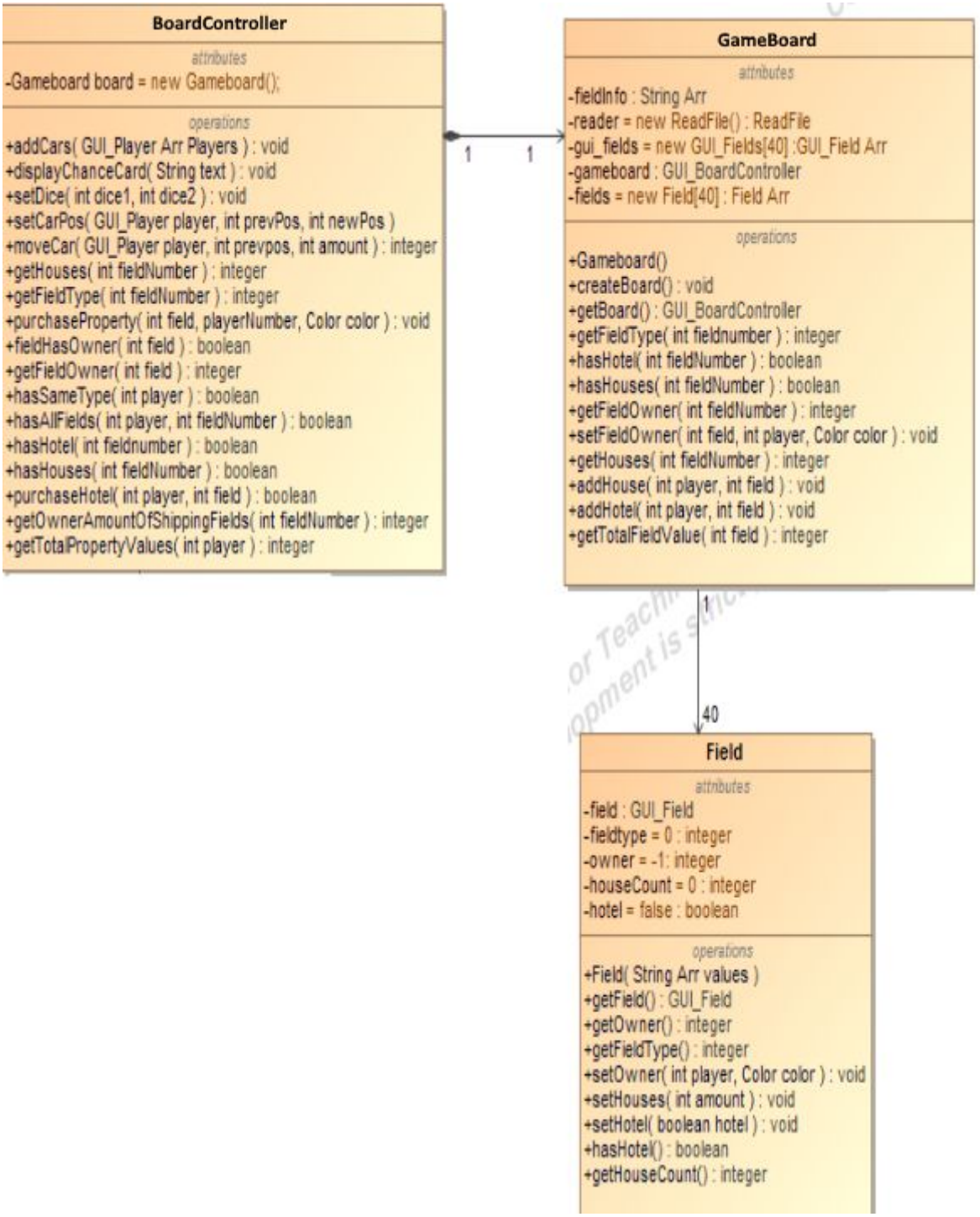


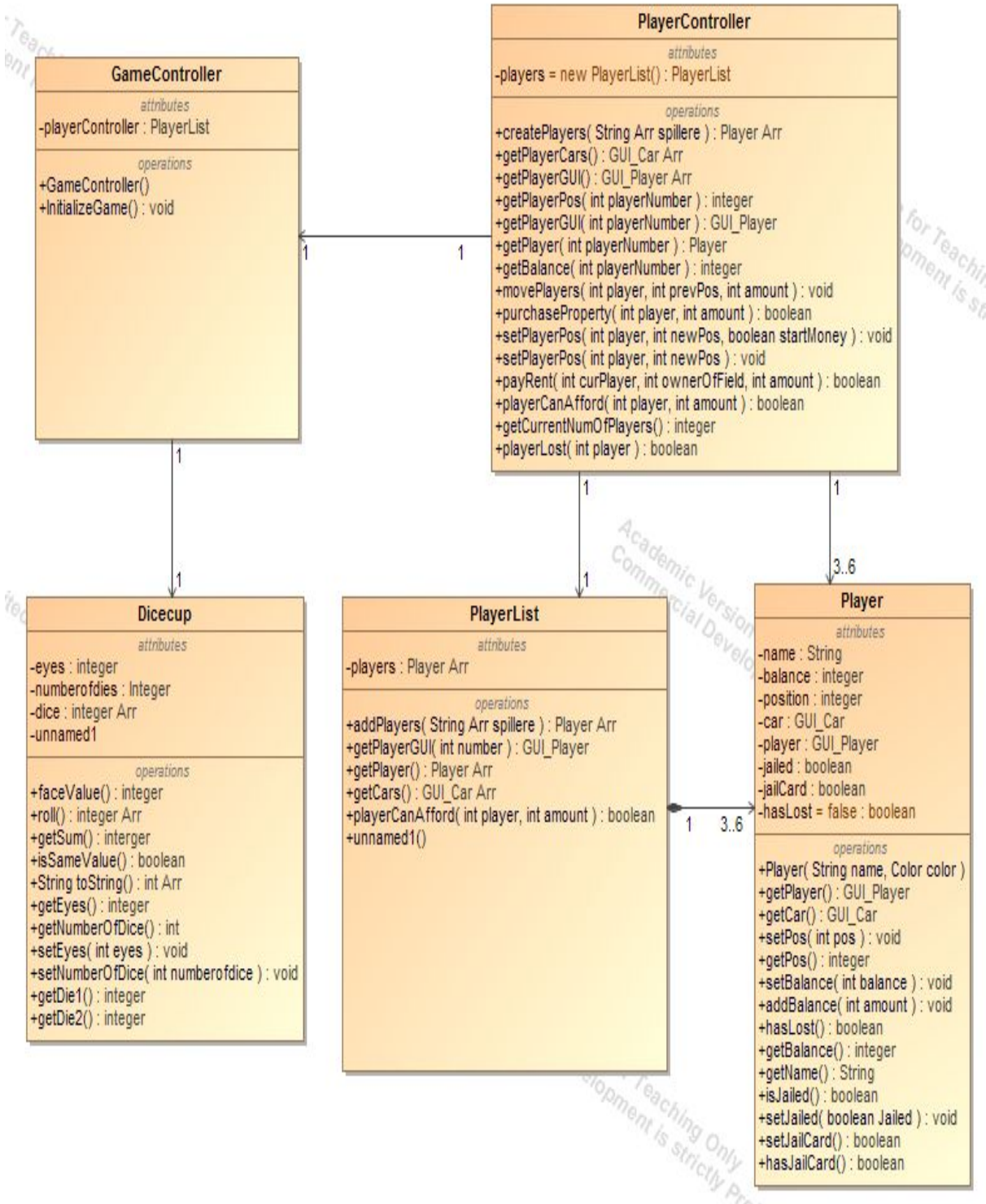
På ovenstående diagram er Playercontrolleren illustreret, i denne controller bliver der holdt styr på de forskellige players. Attributterne i denne klasser sker meget i baggrunden men ikke desto mindre vigtig for programmet. Det er blandt andet her saldoen på de forskellige spillere bliver kontrolleret.

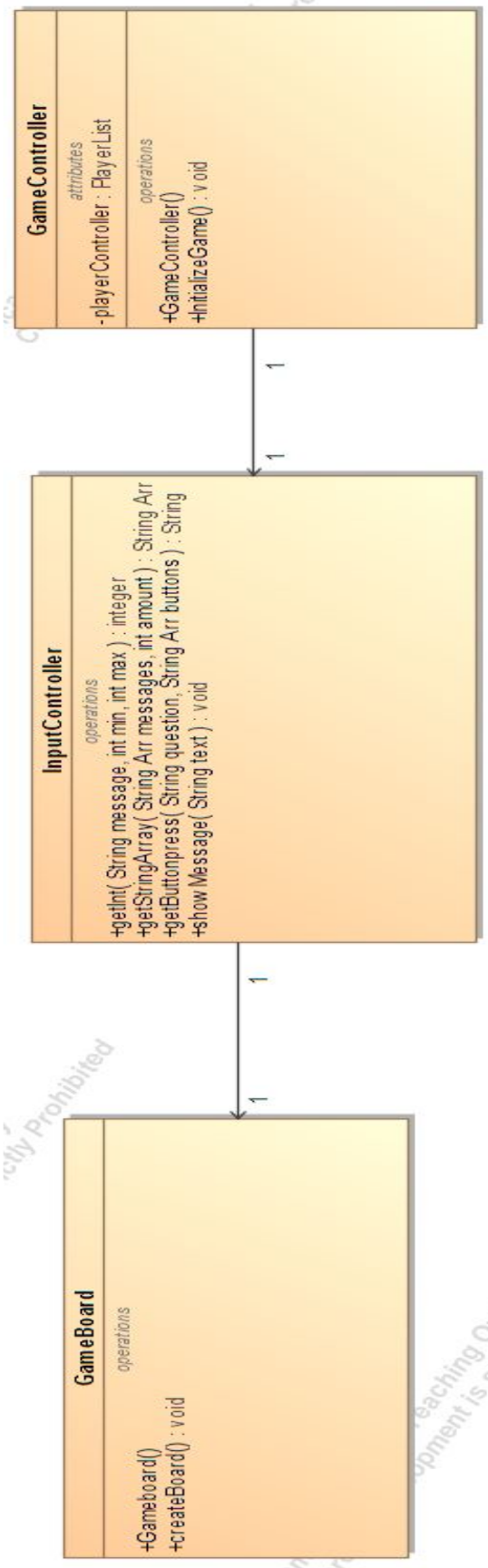
Design Class Diagram

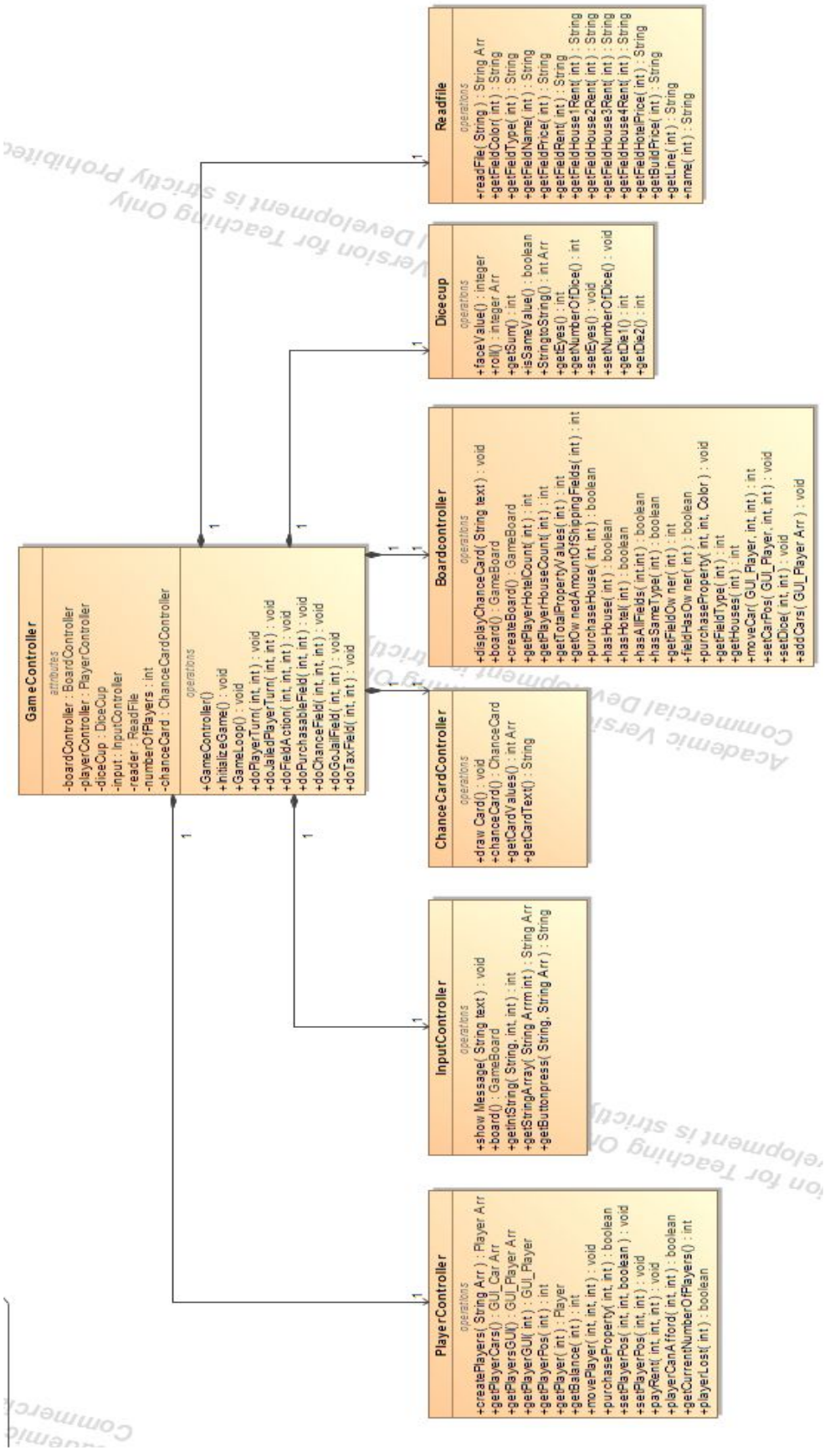
Design klasse diagrammerne viser hvordan hele matador-spillet virker. De klasser, attributter og metoder der optræder i Java koden, er baseret på disse diagrammer.




BoardController
attributes
-Gameboard board = new Gameboard();
operations
+addCars(GUI_Player Arr Players) : void
+displayChanceCard(String text) : void
+setDice(int dice1, int dice2) : void
+setCarPos(GUI_Player player, int prevPos, int newPos)
+moveCar(GUI_Player player, int prevpos, int amount) : integer
+getHouses(int fieldNumber) : integer
+getFieldType(int fieldNumber) : integer
+purchaseProperty(int field, playerNumber, Color color) : void
+fieldHasOwner(int field) : boolean
+getFieldOwner(int field) : integer
+hasSameType(int player) : boolean
+hasAllFields(int player, int fieldNumber) : boolean
+hasHotel(int fieldnumber) : boolean
+hasHouses(int fieldNumber) : boolean
+purchaseHotel(int player, int field) : boolean
+getOwnerAmountOfShippingFields(int fieldNumber) : integer
+getTotalPropertyValues(int player) : integer
1
1
GameBoard
attributes
-fieldInfo : String Arr
-reader = new ReadFile() : ReadFile
-gui_fields = new GUI_Fields[40] :GUI_Field Arr
-gameboard : GUI_BoardController
-fields = new Field[40] : Field Arr
operations
+Gameboard()
+createBoard() : void
+getBoard() : GUI_BoardController
+getFieldType(int fieldnumber) : integer
+hasHotel(int fieldNumber) : boolean
+hasHouses(int fieldNumber) : boolean
+getFieldOwner(int fieldNumber) : integer
+setFieldOwner(int field, int player, Color color) : void
+getHouses(int fieldNumber) : integer
+addHouse(int player, int field) : void
+addHotel(int player, int field) : void
+getTotalFieldValue(int field) : integer
1
40
Field
attributes
-field : GUI_Field
-fieldtype = 0 : integer
-owner = -1: integer
-houseCount = 0 : integer
-hotel = false : boolean
operations
+Field(String Arr values)
+getField() : GUI_Field
+getOwner() : integer
+getFieldType() : integer
+setOwner(int player, Color color) : void
+setHouses(int amount) : void
+setHotel(boolean hotel) : void
+hasHotel() : boolean
+getHouseCount() : integer

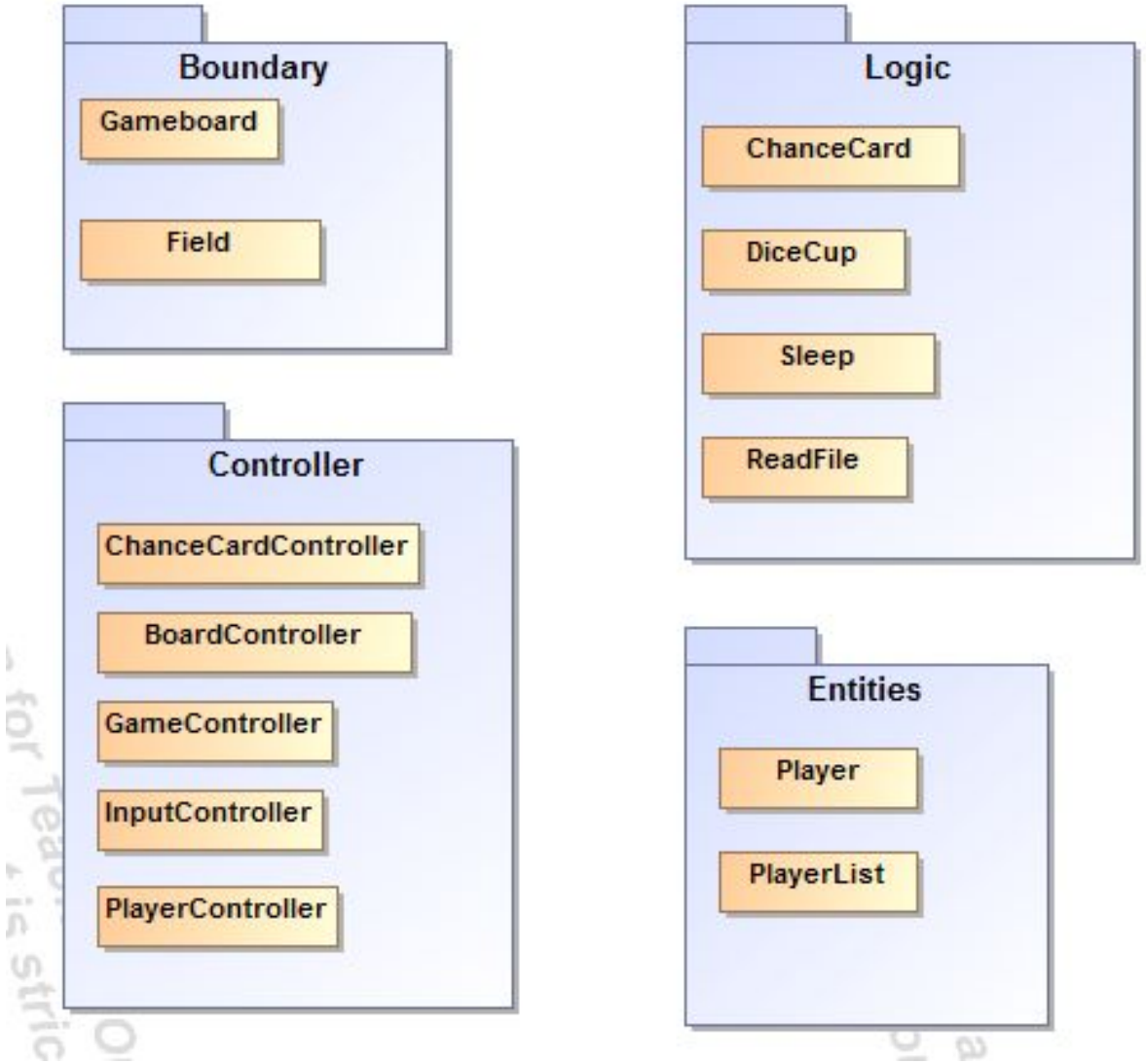

PlayerController
attributes
-players = new PlayerList() : PlayerList
operations
+createPlayers(String Arr spillere) : Player Arr
+getPlayerCars() : GUI_Car Arr
+getPlayerGUI() : GUI_Player Arr
+getPlayerPos(int playerNumber) : integer
+getPlayerGUI(int playerNumber) : GUI_Player
+getPlayer(int playerNumber) : Player
+getBalance(int playerNumber) : integer
+movePlayers(int player, int prevPos, int amount) : void
+purchaseProperty(int player, int amount) : boolean
+setPlayerPos(int player, int newPos, boolean startMoney) : void
+setPlayerPos(int player, int newPos) : void
+payRent(int curPlayer, int ownerOfField, int amount) : boolean
+playerCanAfford(int player, int amount) : boolean
+getCurrentNumOfPlayers() : integer
+playerLost(int player) : boolean
GameController
attributes
-playerController : PlayerList
operations
+GameController()
+InitializeGame() : void
Dicecup
attributes
-eyes : integer
-numberofdies : Integer
-dice : integer Arr
-unnamed1
operations
+faceValue() : integer
+roll() : integer Arr
+getSum() : interger
+isSameValue() : boolean
+String toString() : int Arr
+getEyes() : integer
+getNumberOfDice() : int
+setEyes(int eyes) : void
+setNumberOfDice(int numberofdice) : void
+getDie1() : integer
+getDie2() : integer
PlayerList
attributes
-players : Player Arr
operations
+addPlayers(String Arr spillere) : Player Arr
+getPlayerGUI(int number) : GUI_Player
+getPlayer() : Player Arr
+getCars() : GUI_Car Arr
+playerCanAfford(int player, int amount) : boolean
+unnamed1()
Player
attributes
-name : String
-balance : integer
-position : integer
-car : GUI_Car
-player : GUI_Player
-jailed : boolean
-jailCard : boolean
-hasLost = false : boolean
operations
+Player(String name, Color color)
+getPlayer() : GUI_Player
+getCar() : GUI_Car
+setPos(int pos) : void
+getPos() : integer
+setBalance(int balance) : void
+addBalance(int amount) : void
+hasLost() : boolean
+getBalance() : integer
+getName() : String
+isJailed() : boolean
+setJailed(boolean Jailed) : void
+setJailCard() : boolean
+hasJailCard() : boolean
1
1
1
1
1
1
1
1
1
3..6
1
3..6


GameController
attributes
-playerController : PlayerList
operations
+GameController()
+initializeGame() : void
1
1
InputController
operations
+getInt(String message, int min, int max) : integer
+getStringArray(String Arr messages, int amount) : String Arr
+getButtonpress(String question, String Arr buttons) : String
+show Message(String text) : void
1
1
GameBoard
operations
+Gameboard()
+createBoard() : void


GameController
attributes
-boardController : BoardController
-playerController : PlayerController
-diceCup : DiceCup
-input : InputController
-reader : ReadFile
-numberOfPlayers : int
-chanceCard : ChanceCardController
operations
+GameController()
+initializeGame() : void
+GameLoop() : void
+doPlayerTurn(int, int) : void
+doJailedPlayerTurn(int, int) : void
+doFieldAction(int, int, int) : void
+doPurchasableField(int, int) : void
+doChanceField(int, int, int) : void
+doGoJailField(int, int) : void
+doTaxField(int, int) : void
Readfile
operations
+readFile(String) : String Arr
+getFieldColor(int) : String
+getFieldType(int) : String
+getFieldName(int) : String
+getFieldPrice(int) : String
+getFieldRent(int) : String
+getFieldHouse1Rent(int) : String
+getFieldHouse2Rent(int) : String
+getFieldHouse3Rent(int) : String
+getFieldHouse4Rent(int) : String
+getFieldHotelPrice(int) : String
+getBuildPrice(int) : String
+getLine(int) : String
+name(int) : String
Dicecup
operations
+faceValue() : integer
+roll() : integer Arr
+getSum() : int
+isSameValue() : boolean
+StringtoString() : int Arr
+getEyes() : int
+getNumberOfDice() : int
+setEyes() : void
+setNumberOfDice() : void
+getDie1() : int
+getDie2() : int
Boardcontroller
operations
+displayChanceCard(String text) : void
+board() : GameBoard
+createBoard() : GameBoard
+getPlayerHotelCount(int) : int
+getPlayerHouseCount(int) : int
+getTotalPropertyValues(int) : int
+getOwnedAmountOfShippingFields(int) : int
+purchaseHouse(int, int) : boolean
+hasHouse(int) : boolean
+hasHotel(int) : boolean
+hasAllFields(int,int) : boolean
+hasSameType(int) : boolean
+getFieldOwner(int) : int
+fieldHasOwner(int) : boolean
+purchaseProperty(int, int, Color) : void
+getFieldType(int) : int
+getHouses(int) : int
+moveCar(GUI_Player, int, int) : int
+setCarPos(GUI_Player, int, int) : void
+setDice(int, int) : void
+addCars(GUI_Player Arr) : void
ChanceCardController
operations
+draw Card() : void
+chanceCard() : ChanceCard
+getCardValues() : int Arr
+getCardText() : String
InputController
operations
+show Message(String text) : void
+board() : GameBoard
+getInt(String, int, int) : int
+getStringArray(String Arrm int) : String Arr
+getButtonpress(String, String Arr) : String
PlayerController
operations
+createPlayers(String Arr) : Player Arr
+getPlayerCars() : GUI_Car Arr
+getPlayersGUI() : GUI_Player Arr
+getPlayerGUI(int) : GUI_Player
+getPlayerPos(int) : int
+getPlayer(int) : Player
+getBalance(int) : int
+movePlayer(int, int, int) : void
+purchaseProperty(int, int) : boolean
+setPlayerPos(int, int boolean) : void
+setPlayerPos(int, int) : void
+payRent(int, int, int) : void
+playerCanAfford(int, int) : boolean
+getCurrentNumberOfPlayers() : int
+playerLost(int) : boolean
1

Pakkediagram

I design- og implementeringsfasen har vi lavet en opdeling af vores klasser i "Packages". I hver pakke lægger klasser som har et stort samspil mellem hinanden og skal udføre bestemte opgaver inden for samme ansvarsområde. Disse pakker giver struktur i koden og et større overblik.



Beskrivelse af anvendte GRASP-patterns

High Cohesion

Vores kode består af klasser der hver især står for ét veldefineret ansvarsområde. Vi opnår High cohesion, da klasserne kun har et ansvar og ikke står af op til flere forskellige opgaver. Eksempelvis står DiceCupen kun for at rulle med to terninger og give informationen videre og ChanceCard indeholder kun information om spillets chancekort. Dette gør koden mere overskuelig og meget lettere at teste de individuelle klasser. Dette sammen med Low Coupling er de to patterns vi har arbejdet mest med at opretholde.

Low Coupling

Selvom hver klasse står for et enkelt område, afhænger de fleste af dem også af andre klasser. Også klasser som har et andet ansvar. Den lave kobling i koden, forstås som hver klasses association med en anden. Fra vores design klasse diagram, kan vi se at langt de fleste klasser kun har association til 1-2 andre klasser, hvilket skabet den lave kobling. En klasse skal være så uafhængig som muligt.

Controller

Vi gør stor brug af controllere i vores program, GameControlleren som styre det overordenede program og har kendskab til de forskellige controllere som selv har en specifikation. En af vores controllere hedder PlayerController som holder styr på spillerne, vi har en anden controller der hedder BoardController som holder styr på det der sker på brættet.

Information Expert

For at klasserne kan løse deres opgave, skal de også have den nødvendige information. Information Expert handler om at uddele opgaverne, til de klasser der har den nødvendige information. Eksempelvis har vores PlayerController alt information vedrørende spillerne. Derfor giver det mest mening at PlayerController har til opgave at holde styr på hver spillers saldo, ejendomme og hver spillers brik på boardet.

Creator

Et Creator pattern kan ses i vores Player- og PlayerList klasse. De to klasser arbejder meget tæt sammen, men PlayerList klassen kan ikke stå alene. For at PlayerList klassen overhovedet kan eksistere, skal den kende til et array af Players. Player klassen er altså ansvarlig for skabelse af PlayerList klassen.+

Implementering

Kodeeksempler

GameLoop()

```
private void GameLoop() {
    int curPlayer = 0;
    int prevPos;

    while (true) {
        if (curPlayer >= numberOfPlayers) curPlayer = 0;
        prevPos = playerController.getPlayerPos(curPlayer);

        if(playerController.getPlayer(curPlayer).isJailed()) {
            doJailedPlayerTurn(curPlayer, prevPos);
        } else {
            doPlayerTurn(curPlayer, prevPos);
        }

        if (playerController.playerLost(curPlayer)) {
            break;
        }

        if (!diceCup.isSameValue()) {
            curPlayer++;
        }
    }
```

Vores GameLoop metode er spillet i sin mest simple form. Denne metode kalder på andre metoder og det er dem, som laver alt arbejdet.
Vi kører rundt i en løkke, som altid er true. Dette betyder at hvis vi gerne vil ud af denne løkke, skal vi break'e ud; dette kommer vi til.
Vi kører rundt mellem de forskellige spillere ved først at sætte curPlayer (current player) til 0. Den vil nu kører spillet igennem for den spiller og som vi kan se nederst, skriver vi, at hvis der IKKE bliver slået to ens, skal spillerturen gives videre og vi lægger derfor én til værdien curPlayer.
Øverst i loopet tjekker vi om vi har været alle spillerne igennem. Hvis vi har dette, sætter vi curPlayer til 0 igen ellers er det blot den næste spillers tur.

Vi gemmer spillerens position som prevPos (previous position), da dette er positionen før spilleren har slået.
Vi tjekker da om spilleren er i fængsel, hvis dette er tilfældet kalder vi doJailedPlayerTurn med parametrene curPlayer og prevPos. Hvis dette ikke er tilfældet kører vi metoden doPlayerTurn med de samme to parametre.
Nu tjekker vi, om vi er færdige med spillet. Vi kigger nemlig på om spilleren har tabt og hvis dette er sandt break'er vi ud af while loopet. Hvis det er falsk, går vi videre til næste spiller.

doJailedPlayerTurn()

```
for (int i = 0; i < 3; i++) {
    input.getButtonpress( question: "Spiller: " + playerController.getPlayerGUI(player).getName() +
            "\nDu er i fængsel, slå 2 ens for at komme ud.\nForsøg " + (i+1) + "/3", new String[]{"kast"});
    diceCup.roll();
    boardController.setDice(diceCup.getDie1(), diceCup.getDie2());

    if (diceCup.isSameValue()) {
        playerController.getPlayer(player).setJailed(false);
        fieldNumber = boardController.moveCar(playerController.getPlayerGUI(player), prevPos, diceCup.getSum());
        playerController.movePlayer(player, prevPos, diceCup.getSum());

        doFieldAction(player, prevPos, fieldNumber);
        break;
    }
}
```

Billedet kan også ses i bilag 1

Her ses et udklip af doJailedPlayerTurn metoden. Det er her spilleren kan få lov at komme ud af fængslet, hvis personen slå to ens med terningerne på ét af de tre forsøg, vedkommende har.
Vi laver et for-loop, som kører tre gange. Her skriver vi ud til spilleren at personen skal slå to ens med terningerne og vi skriver hvilken af de tre ture, spilleren er i gang med.
Vi tjekker så om de to terninger har samme værdi ved hjælp af metoden isSameValue fra vores diceCup klasse. Hvis dette er false, går vi ikke ind i if-statementet, men hvis spilleren formår at slå to ens, sætter vi player variablen "jailed" til false og herefter flytter vi spilleren det antal, som blev slået.
Til slut kalder vi vores metode doFieldAction med tre parametre. Denne metode finder ud af hvad der skal ske ud fra feltet, der landes på. Herefter break'er vi ud af for loopet.

moveCar() - BoardController

```
public int moveCar(GUI_Player player, int prevpos, int amount) {
    boolean moving = true;
    int movepos = prevpos + 1;
    int newpos = prevpos + amount;
    movepos %= 40;
    newpos %= 40;

    while(moving) {
        board.getBoard().getFields()[prevpos].setCar(player,  hasCar: false);
        board.getBoard().getFields()[movepos].setCar(player,  hasCar: true);
        prevpos++;
        prevpos %= 40;
        movepos = prevpos + 1;
        movepos %= 40;
        int speed = 200;
        //acceleration
        int newSpeed = (int)(speed - (amount * 1.4));
        sleep(newSpeed);

        if (prevpos == newpos) {
            moving = false;
        }
    }

    return newpos + 1;
}
```

I vores BoardController klasse findes denne metode. Når vi kalder metoden, flytter bilen sig ét felt af gangen op til det felt, som spilleren skal ende med ud fra terningeøjnene. Dette får det til at ligne virkeligheden mere end hvis man blot flyttede spilleren det antal øjne, som blev slået.
Vi starter med at lave nogle variable movepos og newpos, hvor movepos er det næste felt i forhold til det felt, som spilleren står på og newpos er det felt, som spilleren skal ende med at stå på. Herefter moduluser vi begge tal med 40, da vores felter er nul-indexerede. Det har den fordel, at spilleren flytter sig i ring på pladen. F.eks. hvis spilleren står på felt 38 og slå seks med terningerne, så kommer spilleren op på felt 44, men da dette ikke findes, tager vi resten af tallet, som ikke går op i vores heltalsdivision. Vi får da at 40 går op i 44 én gang og der er 4 til rest. Spilleren lander altså på felt 4.
Herefter laver vi en int, som vi kalder speed. Dette er grundet, at vi godt ville have, at der var forskel i hvor hurtigt en brik rykkede sig alt efter hvor meget man slog med terningerne.

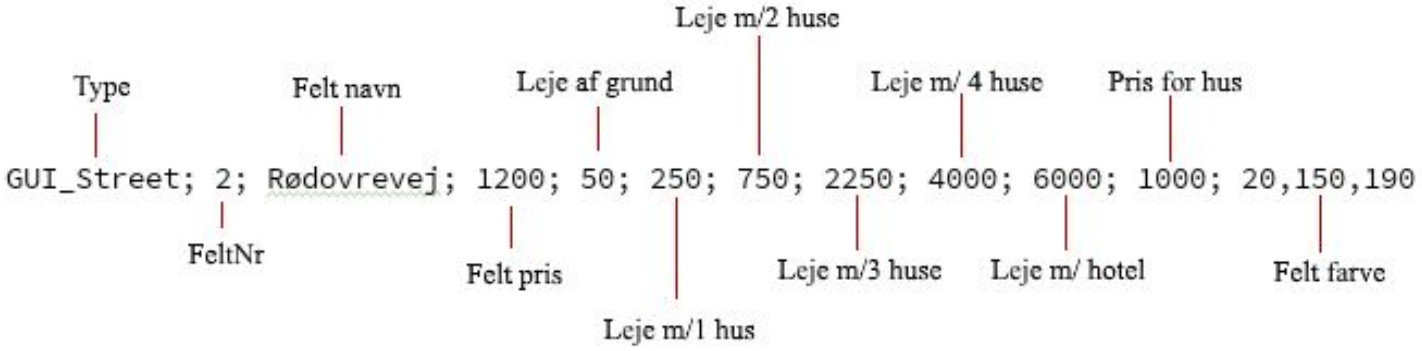
Vi laver derfor en ny variabel, som vi kalder newSpeed, hvor vi laver en udregning, der giver os en acceleration ud for "amount", som er summen af de to terninger.
Dette tal lægger vi ind i en sleep metode og det er den, der bestemmer hvor hurtigt brikken rykker. Når vi er færdig med sleep metoden, som venter x-antal millisekunder kører vi while loopet igen, hvis vores position ikke er lig den position, vi gerne vil ende med, og her fjerner vi brikken fra hvor den før stod og sætter den på det næste felt. Dette synes vi giver en god visuel effekt af brikken.

Tanken bag tekstfilerne

```
GUI_Start; 1; Start; 4000; 255,0,0
GUI_Street; 2; Rødovrevej; 1200; 50; 250; 750; 2250; 4000; 6000; 1000; 20,150,190
GUI_Chance; 3; Prøv lykken;
GUI_Street; 4; Hvidovrevej; 1200; 50; 250; 750; 2250; 4000; 6000; 1000; 20,150,190
```

Her ses et udklip af vores .txt fil "fieldInfo.txt". Denne fil bruges til at hente oplysninger fra, som bruges inde i selve koden. På den måde slipper man for at "hardcode" de forskellige informationer om de forskellige felter.
Inde i koden har vi brugt String.split metoden til at splitte linjerne op ved "; " og ud fra det får vi et String array. Ud fra dette kan vi hente felttypen på et givent felt ved at tage index 0 i det array, vi fik ved at splitte String'en op. Index 1 bruges til felt numrene og index 2 er navnet på feltet. Ved at tage den sidste String i vores splitArray kan vi få farven til et givent felt. Dette fås ved at sige splitArray[splitArray.length-1].

På vores GUI_Street's (felter, man kan købe og sætte huse på) betyder de forskellige tal følgende:



Test af kode med J-Unit

De forskellige testcases

drawCard()

```
@Test
public void drawCard() {
    //lavet et CardSet og blander det.
    chanceCard.makeCardSet();
    chanceCard.shuffleCard();
    String topCard = chanceCard.getCardSet()[0];
    chanceCard.drawCard();
    String bottonCard = chanceCard.getCardSet()[chanceCard.getCardSet().length-1];
    //tjekker om kortet er blevet langt nederst i bunken.
    assertTrue(topCard.equals(bottonCard));
}
```

Vores drawCard metode skal kunne trække det øverste kort i bunken og lægge det nederst bagefter.
Vi starter med at lave et kortsæt og bagefter blander vi det.
Herefter gemmer vi det øverste kort i bunken, som gerne skulle være det, som skal trækkes og lægges nederst i bunken.
Så kalder vi drawCard metoden og bagefter gemmer vi så det nederste kort.
Vi tester ved at kigge på om vores øverste kort - før vi træk - er det samme som det nederste kort efter vi har trukket.

addBalance()

```
@Test
public void addBalance() {
    int startBalance = player.getBalance();
    int moneyToAdd = 100;
    player.addBalance(moneyToAdd);
    assertTrue( condition: player.getBalance() == startBalance + moneyToAdd);
}
```

Vores addBalance metode skal kunne lægge penge til en forhenværende pengebeholdning.
Vi starter med at gemme spillerens balance som den er lige nu.
Nu skriver vi et beløb, som vi gerne vil lægge til.
Bruger vores addBalance metode med de penge, vi vil lægge til, som parameter.
Vi tester ved at se om spillerens balance er lig med vores startBalance + de penge vil ville lægge til. Alle vores test består.

Brugertest - tænke-højt-test

En voksen, kvindelig bruger, som kender til tænke-højt-test og teorien bag denne, men ikke til programmering.

Ting bruger sagde

Hvor ved man hvor mange man skal være?
Vi har 30000 hver.
Man kan ikke se regler nogen steder. Der står ikke hvor meget man får over start osv.
Kaster og den rykker selv.
Fedt hvis der stod. "Du har nu købt denne grund."
Det er godt, at farverne på de købte grunde er den samme som bilernes.
Stavefejl i betale.
Så man har ekstra slag efter 2 ens? Det står ikke nogen steder.
Ikke så godt at terningerne er over tekst.
Man får ikke at vide, at man passere start.
Tekst med at man selv ejer felt er ikke så godt. Skal jeg give mig selv penge?
Skriv "du har ingen udgift" til chancekort med betal hus og hotel.
Stavefejl i "betale"
Ikke spørge om man vil bygge, når man har hotel
Sig at man ikke har råd til huse, hvis man ikke har nok penge
Den kommer ikke op med en besked med at man ikke har råd til at købe en grund

Bugs, vi fandt

Flyt bil, når chance kort i fængsel. Spawner fra fængsel.
Ser rådhuspladsen som chancekort, når man trækker gå til rådhusplasen
Ryk tre felter frem fjerner ikke bilen fra pos den stod på. (spillet crasher) der kommer ikke nogen knap op.
Den kommer frem et nyt chancekort, når man tager til grønningen fra chancekort
Stavefejl i chancekort "fængselskort".
Når man får et chancekort, hvor man skal tage hen til et andet felt, trækker den et nyt chancekort.
Huse overruler ikke ejer af alle 3 grunde. Hotel virker selvom man har 2 hoteller.
Trækker ikke nogen penge, hvis man lander på hotel og man ikke har råd. Man burde tabe!
Den skal ikke tjekke om man har råd til at betale leje.

Valg af tests

Vi har valgt at teste vores logik, dvs. DiceCup, chanceCard og ReadFile metoder med unit tests, da det giver et hurtigt overblik over ens logik i spillet. Man kan med unit test hurtigt teste enkelte dele meget nemt.
Den måde, vi har valgt at teste spillet i sin helhed, var ved en brugertest. Vi fik i denne test samlet en masse småfejl sammen, som har hobet sig op hen af vejen. Vi fik skrevet dem ned, hvorefter vi talte sammen i gruppen om hvordan vi bedst kunne løse disse. Nogle af bugsne var relativ nemme at løse, da det hovedsageligt var småfejl og værdier, som blev overført forkert mellem de forskellige metoder.
En tænke-højt-test giver en præcis virkeligheds kopi af en usecase, nemlig at spille spillet. Dog er det vigtigt, at det ikke er en person, som har været med til at skrive koden, som også tester det, da disse kan være forudindtaget eller specifikt vil undgå at gøre en bestemt ting. En udefrakommende person vil gøre hvad der falder mest naturligt for ham eller hende. Dette vil i de fleste tilfælde ende ud i, at man finde nogle fejl, som man ellers ikke lige ville havde fundet ved blot at teste én eller flere mindre dele af gangen. Dette er altså en test af hele systemet.

Projektforløb

Logbog

07/1 - 2019
I dag har vi lavet de forskellige krav til produktet og snakket om hvad vi gerne vil lægge vægt på i programmet og hvad vi ser som mindre vigtigt. Vi har også lavet en tidsplan.
I morgen vil vi lave alle de forskellige diagrammer. Vi ser også på use cases, navneordsanalyse og PI matrix.

08/1 - 2019
I dag har vi lavet forskellige use cases og lavet klassediagram. Vi har også lavet fully dressed af et af use case'ne. Vi fik også lavet en PI matrix.
I morgen skal vi fremvise, hvad vi har lavet indtil nu. Vi vil lave system sekvens- og sekvensdiagram og design klassediagram. Måske vil vi også starte på programmeringsdelen.

09/1 - 2019
I dag har vi præsenteret vores prioriteret kravliste, samt Use Cases for vores kunde og tekniske projektleder. Yderligere har vi fået færdiggjort System Sekvens Diagrammerne og gjort vores GUI klar til Implementerings-fasen.
I morgen skal vi blive færdige med Design Sekvens Diagrammet, samt Design Klasse Diagrammerne. Vi tænker også at begynde på Implementeringen og få klasser fra tidligere

CDIO opgaver overført i vores nye sourcecode. Vi genbruger terningerne fra CDIO 2 og opstarten af Junior matadorspillet fra CDIO 3.

10/1 - 2019
I dag har vi lavet en lille smule om på nogle af vores diagrammer efter samtalen med hjælpelærer. Vi er også begyndt at få sat diceCup klassen ind og selve pladen.
I morgen vil vi skrive noget mere kode.

11/1 - 2019
Vi har i dag snakket mere om hvordan vi vil implementere de forskellige dele af spillet og hvem der skal lave hvad i weekenden.
På mandag skulle vi gerne have diceCup klassen færdig. Vi vil også lave nogle flere test af de forskellige klasser.

14/1 - 2019
I dag var tre fra gruppen syge. Vi fik lavet lidt på Player klassen og lavet nogle forskellige metoder, som kan bruges, når vi giver navne og priser på de forskellige felter. Vi fik også lavet nogle forskellige tests.
I morgen vil vi prøve at få lavet meget af spil logikken færdig. Forhåbentlig få lavet så spillerne kan rykke rundt på pladen.

15/1 - 2019
I dag har vi lavet selve brættet. Vi har lavet farverne og navne og alt på de forskellige felter - så det ser godt ud!
Vi har også fået spillerne til at kunne bevæge sig på pladen og få penge over start. Vi har nu i alt otte tests.
I morgen skal vi fremlægge, det vi har indtil videre.
Vi vil også kigge på chance kort.

16/1 - 2019
I dag har vi haft statusmøde omkring spillet, hvor vi viste vores prototype frem. Kernen af spillet virker godt, men vi mangler ét “Must Have” krav, samt et par “Should Have” krav. Vi fik præsenteret vores plan for de sidste dage i 3-ugers perioden. Vi fik konstruktiv kritik af vores plan og vi fik ændret i planen efter mødet. Efter mødet arbejdede vi videre med rapportskrivning, samt implementation af chancekort, fængslet og opkøbning af grunde. Når vi er færdige med hele implementeringen, skal vi beskrive koden og klasserne i rapporten.
I morgen skal vi hovedsageligt arbejde videre med koden.

17/1 - 2019
I dag har vi arbejdet videre med koden og fået rettet nogle diagrammer til.
I morgen arbejder vi videre med implementering af koden.

18/1 - 2019
I dag valgte vi at arbejde hjemmefra og snakke sammen over discord. Igen i dag arbejdede vi med koden. Vi fik struktureret koden lidt bedre inde i vores gameController.

19/1 - 2019
I dag har vi lukket spillet og implementeret de sidste features, som vi gerne ville have med. Vi er tilfredse med den færdige version, trods nogle kompromiser. Vi fik ikke sluttet spillet, som vi allerhelst ville, men vi mener selv, at vi har fundet den anden bedste måde at gøre det på.
I morgen skal vi skrive nogle kommentarer til koden, som vi mangler og ellers skal vi lægge sidste hånd på rapporten.

20/1 - 2019
I dag har vi skrevet flere kommentarer i koden og vi har skrevet videre på rapporten.
Ydermere har vi også lavet en brugertest og fixed forskellige bugs, som var i spillet.
Vi aflevere projektet i dag.

Konfiguration

Udviklingsplatform

Softwaren der blev brugt under udviklingen var:
IntelliJ
Windows 10
Windows 8
Git
Maven
GUI

Import fra Git repository[2]

Når man vil importere filerne fra et 'Git repository' ind i et udviklerværktøj som IntelliJ for at arbejde videre med det, gøres dette nemmest på følgende måde:

1. **Kopiere linket til 'Git repository' på GitHub.**
2. **Åbn IntelliJ** (hvis man har et projekt åbent kan man fortsætte til step 3).
    a. Læs videre her, hvis et projekt ikke allerede er åbnet i IntelliJ .
    b. Klik på "Check out from Version Control"
    c. Klik på Git
    d. Kopiér linket ind i "URL" boksen og gem det et sted, du vil have det lokalt.
    e. Klik på "clone"
3. **Hvis du har et projekt åbent, så læs videre herfra:**
4. **Klik på "VCS" i baren i toppen.**
5. **Klik på "Check-out from Version Control".**
6. **Klik på "Git".**
7. **Kopiér linket ind og vælg et sted, den skal gemme projektet lokalt.**
8. **Klik på "clone".**

Nu kan du commit, push og pull fra det 'Git repository'.

Konklusion

Vi kan konkludere at vi fik implementeret 15 ud af 30 krav, hvoraf vi fik implementeret alle 'must haves' og 'should haves' i vores MoSCoW. Kunden havde et ønske om at vi prioriteret programmet fungere højere, end det at få implementeret alle kravene. Dette har vi selvfølgelig overholdt og derfor brugt en masse tid på at teste for fejl. Udover programmet har vi selv været meget velfungerende. Vi er meget tilfredse med resultatet og arbejdsprocessen, og på trods af en smule sygdom, er vi kommet godt gennem hele projektet.

[2] Dette er taget fra vores eget CDIO2 projekt.

Bilag

Bilag 1 - doJailedPlayerTurn()

```
for (int i = 0; i < 3; i++) {
    input.getButtonPress( question: "Spiller: " + playerController.getPlayerGUI(player).getName() +
            "\nDu er i fængsel, slå 2 ens for at komme ud.\nForsøg " + (i+1) + "/3", new String[]{"kast"});
    diceCup.roll();
    boardController.setDice(diceCup.getDie1(), diceCup.getDie2());

    if (diceCup.isSameValue()) {
        playerController.getPlayer(player).setJailed(false);
        fieldNumber = boardController.moveCar(playerController.getPlayerGUI(player), prevPos, diceCup.getSum());
        playerController.movePlayer(player, prevPos, diceCup.getSum());

        doFieldAction(player, prevPos, fieldNumber);
        break;
    }
}
```