PiSU

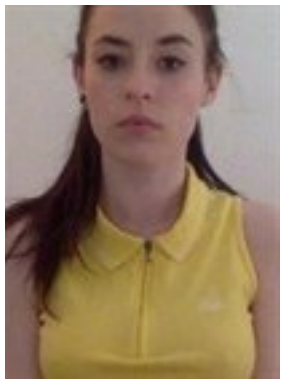
02362 Projekt i softwareudvikling
02327 Indledende databaser og database programmering

Gunn Hentze
s145494 (GH)

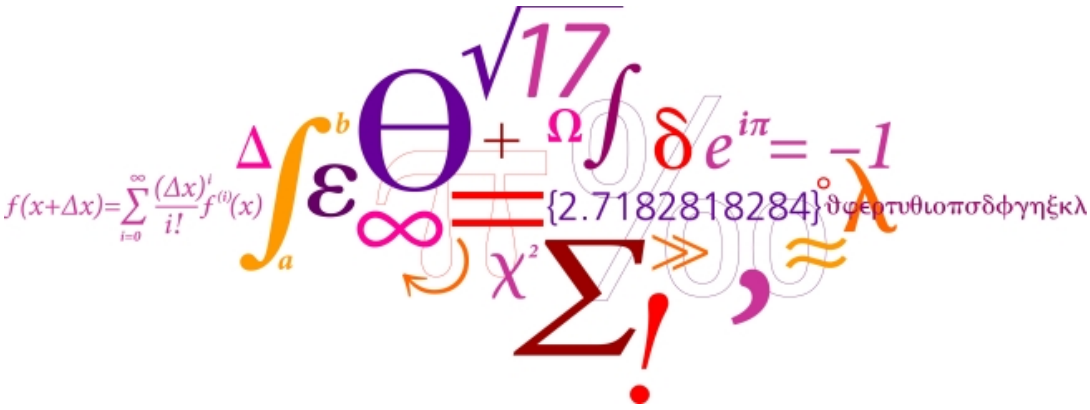
Oliver Schultz
s175113 (OS)

Janus Olsen
s175129 (JO)

Christian Høj
s175125 (CH)



Josephine Mellin
s163313 (JM)



1. Introduktion

I kurserne Projekt i softwareudvikling og Indledende databaser og database programmering, går opgaven for projektet ud på at udvikle et virkelighedens Matadorspil for voksne. Igennem rapporten vil de forskellige punkter beskrives, som vores arbejde har været omkring under udviklingen af projektet. Herunder analysen delen, hvor der er lokaliseret de funktionelle og ikke-funktionelle krav spillet har; hvad programmet skal kunne ud fra brugerens synspunkt. Disse krav til spillet er beskrevet gennem et væld at artifacts, som hver især griber kravene an på forskellig vis.

Under designdelen af rapporten beskrives der hvordan der ønskes softwaren skal fungere med henblik på analysen. Her er der ligeledes udarbejdet artifacts, som både går i dybden med programmets funktionalitet og virkning f.eks. gennem klassediagram, hvori klasser, deres attributter, metoder og ansvar visualiseres (mht software arkitekturen). Ligeledes hvordan der arbejdes med databasen, der skal være behjælpelig for at brugeren kan spille Matador, hvordan værdier og informationer skal gemmes, når Matador spilles. Diverse brugte udviklingsværktøjer og udviklingsprocesser vil også blive gennemgået og diskuteret, ligesom diverse tests vil udføres og gennemgås.

Til sidst vil en afsluttende konklusion for projektet redegøre for de opnåede mål.

2. Problembeskrivelse

Spillets software udvikles i sproget Java, og i programmet Eclipse Jee Oxygen. Spillets software vil indeholde de nødvendige klasser, attributter og metoder for at spillet fungerer optimalt og indenfor spillets regler. Foruden spillets software skal en tilhørende lokal database udvikles i sproget SQL, ved hjælp af programmet MySQL Workbench. Den tilhørende database vil give mulighed for at spillets spillere kan gemme og lukke et spil, og senere åbne det samme spil, fra samme udgangspunkt, fra da det blev lukket. Det færdige spil skal efter bedste evne, efterligne det virkelige Matador spil. For at opnå dette skal følgende funktionelle krav indgå i spillet. Disse krav er identificeret ved hjælp af MoSCoW, som den første del af analysen. Se videre for analysen og relevant artifacts under kravsanalysen (punkt 3).

2.1 MoSCoW @OS

Must have

- En spilleplade i form af en brugergrænseflade.
- Spillerne skal kunne vælge, hvor mange de er til spillet.
- Spillerne skal kunne skrive navne ind.
- Spillepladen skal have 40 felter.
- Antallet af spillere skal kunne variere mellem 3-6.

- Hver spiller skal have en brik.
- Hver spiller skal modtage en startbeholdning på 30.000 kr.
- Spillet skal indeholde 2 terninger.
- Spillet skal indeholder 32 chancekort.
- Hvis en spiller slår 2 af samme slags, modtager vedkommende et ekstra slag. Dette ekstra slag kan udløse yderligere ekstra slag.
- Hvis en spiller slår 2 af samme slags 3 gange i træk, skal vedkommende rykkes i fængsel.
- Hver gang en spiller passerer eller lander på "Start" modtager vedkommende 4.000 kr.
- Hvis en spiller lander på et ikke-købt felt, skal vedkommende have mulighed for at købe det.
- Hvis en spiller ikke køber det felt han/hun lander på, skal alle have mulighed for at købe dette via en auktion.
- Hvis en spiller lander på et felt, ejet af en anden spiller, betaler vedkommende ejeren den felt-bestemte leje. Medmindre ejermand er fængslet.
- Hvis en spiller lander på "Prøve lykken", trækker vedkommende et kort og følger anvisningen.
- Hvis en spiller lander på "De fængsles", går vedkommende direkte til "Fængsel" og modtager ikke startbonus.
- En fængslet spiller skal kunne komme ud på følgende måder: 1) Ved at betale en bøde på 1.000 kr. inden sit slag. 2) Ved at bruge et chancekort. 3) Ved at slå 2 af samme slags, spilleren skal rykke frem det antal øjnene viser, desuden modtager han et ekstra slag.
- En fængslet spiller, må ikke købe ejendomme eller kræve leje, men kan godt være med til at byde under auktion.
- En spiller skal kunne pantsætte sin ejendom, for at modtage lån af banken. Renten er 10 % og betales samtidigt med tilbagebetalingen af lånet. Pantsætningen ophæves efterfølgende.
- En spiller må ikke opkræve leje for en pantsat ejendom.
- Banken giver kun lån mod pantsætningssikkerhed.
- En spiller går bankerot, hvis han skylder mere end han ejer. Pantsatte ejendomme har en værdi på 50 % af købspris.
- En spiller vinder, når de resterende spillere er gået bankerot.
- Programmet skal være tilkoblet en lokal database, som gemme væsentlige værdier såsom spillernavn, pengebalance, opkøbte felter og bilbriksfarven for hver spiller.
- Databasen skal ligeledes være i stand til at gemme flere spil af gangen i det tilfælde af en spiller, indgår i flere spil. Spillet gemmes via oprettelsesdatoen og få datoen for det gemte spil.
- En spiller kan først bygge, når vedkommende ejer en hel gruppe ejendomme.
- Ejendoms-felterne er opdelt i grupper, hvor hver gruppe har en unik farve.

Should have

- En spiller kan købe huse til sin ejendomme, der må højst bygges 4 huse på en grund.
- En spiller kan kunne købe et hotel til sin ejendom, hvis vedkommende har 4 huse stående på grunden. Der må højst bygges 1 hotel på en grund.
- En spiller skal sælge bygninger tilbage til banken, før vedkommende pantsætter sin ejendom. Sælges til 50 % af købsprisen.
- Det skal være tilfældigt, hvilke chancekort, der trækkes fra bunken.
- Ønsker to spillere at der skal ske en byttehandel, skal det aftales indbyrdes og indtastes af spillerne selv, før det effektiviseres.
- Der skal bygges jævnt. Hvis man bygger et hus på en grund, er det ikke tilladt at bygge yderligere på grunden, før man har bygget et hus på de resterende grunde i gruppen.

Could have

- En spiller kan til enhver tid sælge bygninger tilbage til banken til 50 % af købsprisen.
- Spillerne har mulighed for at vælge deres brikfarve.

Won't have (at this time)

- At spillere kun gives mulighed for at købe, sælge og pantsætte, så snart det er deres tur og ikke indimellem. Medmindre dette omhandler byttehandel eller under auktion.
- Hvis der skulle være tid til det, kunne man ændrer i GUI'en således at spillepladen var rund, og tilsvarende den oprindelige spilleplade for Matador.
- Hvis der skulle blive tid, kunne man til GUI'en have lavet et raflebæger, der kunne "holde" de to terninger.
- Der inddrages ikke en "hurtig funktion" til Matadorspillet, hvori der gives to tilfælde skøder og aftale en specifik spilletid.
- En spiller må først købe ejendomme, når vedkommende har passeret start en gang.
- Selvom det indgår i reglerne, at der er et maks antal for huse og hoteller, der er tilgængelige for købt, vil dette ikke inddrages i programmet.

3. Analysen (kravspecifikation)

I analysen identificeres kravene for programmet hovedsageligt, som her be- og gennemarbejdes via artifacts. Dette er et led til at kunne gå igang med softwarenarkitekturen uden større problemer, da det på dette tidspunkt skulle være klart, hvilke koncepter og metoder, der skal indgå.

3.1 FURPS @JM

For at beskrive spillets krav udefra, herunder fra brugerens perspektiv, gøres der brug af FURPS modellen, dog kun i henhold til ikke funktionelle krav, som beskrives gennem usability, reliability, performace og supportability. Her er det punkterne usability og reliabilty som har relevans for en tilfældig bruger af spillet, og som en bruger med rimelighed kan forvente af et virtuelt brætspil.
De resterende punkter performance og supportability er de krav, som kan ses som relevant punkter for hvordan spilles skal gennemføres.

Usability:

- **U1:** Spilleren skal kunne se den værdi terningerne slår.
- **U2:** Spilleren skal kunne se sit navn og sin pengebeholdning.
- **U3:** Spillepladen skal være overskuelig rent visuelt, så man nemt kan se, hvilke felter, der hører sammen.
- **U4:** Spilleren skal kunne se, hvis tur det er, og hvor deres bilbrik befinder sig på brættet.
- **U5:** Programmet skal kunne forklare spillets regler kort og overskueligt, således at det at spillerne ikke er nødsaget til at se spillets manual.
- **U6:** Programmet skal meddele spilleren om de forskellige pengeværdier der ændres, enten trækkes fra eller lægges til den givne spillers konto. Pengebeholdning opdateres efter de opkøbte felter, der landes på.
- **U7:** Programmet skal kunne vise, så snart en grund (property) ejes, ved at omramme feltet i samme farve, som den givne spillers bilbrik.
- **U8:** Programmet skal kunne vise, når der er købt huse eller hotel, ved at sætte på dem på grunden (Real Estate).
- **U9:** Programmet skal kunne meddele spilleren visuelt, om hvilket chancekort han/hun har fået (Prøv lykken).
- **U10:** Det skal være nemt at starte spillet op.
- **U11:** Spillet må ikke kunne crashe.
- **U12:** Spillets regler skal stemme overens med de velkendte regler fra det "virkelige" Matador.

- **U13:** Programmet skal kunne meddele spilleren, om han/hun enten er gået bankerot og er ude af spillet, eller når alle andre spillere er gået bankerot og én spiller bliver udnævnt som vinder.

Reliability:
- **R1:** Det forventes at programmet kan aflæse et vilkårligt felts funktioner.
- **R2:** Det forventes at spillet kun kan spilles, hvis der er 3-6 spillere.
- **R3:** Det forventes at alle spillere starter med en pengebeholdning på 30.000kr.
- **R4:** Det forventes at programmet kan aflæse, hvad end om der skal trækkes penge fra eller lægges til afhængigt af de forskellige felter og deres funktioner.
- **R5:** Det forventes at spillet kan gemmes via en database, og åbnes igen fra samme udgangspunkt.

Performance:
- **P1:** Terningernes værdi skal vises efter 333 millisekunder.
- **P2:** Det forventes at meddelelser til spilleren vises efter 333 millisekunder.
- **P3:** Bilbrikkerne skal rykkes samme antal felter, som terningernes værdi, hurtigt efter et slag.

Supportability:
- **S1:** Spillet skal kunne spille på en normal computer, a la dem fra DTU's databar.
- **S2:** Spillet skal være koblet op til en lokal database.
- **S3:** Brugeren skal have den nyeste version af både eclipse og MySQL Workbench.
- **S4:** Brugeren skal ligeledes have MySQL community server for at kunne bruge spillet.

3.2 Use case @JM

Til de udarbejdede use cases, er der lavet en overordnet, som beskrives her igennem brief og casual metoden. Der er også udarbejdet en fully dressed, som ligger i bilagene. Hertil "zoomes" der ind på de scenarier af spillet, som har flest alternative flows, for at kunne give et overblik af disse. De use cases, som ikke indgår i rapporten, ligger i appendix 1.

Use Case description
ID 1: PlayGame
Brief
3-6 spillere (Player1 - Player6) med hver en forskelligfarvet bil, huse, hoteller og færger, starter med en pengebeholdning på 30.000kr. Der spilles på en spilleplade, som indeholder 40 felter med forskellige muligheder og funktioner; nogle kan købes, sælges og giver mulighed for at trække kort. Rammer en modstander et ejet felt, skal der betales leje til ejermanden af feltet. Beløbet vil variere alt efter, hvor mange felter der ejes af samme type, og hvor mange huse eller hoteller, der er på. Spillet spilles fortsat indtil at alle på nær én spiller er gået bankerot[1].

ID 1: PlayGame
Casual
- Spillerne (Player1-6) opstarter spillet, og indskriver antal af spillere i systemet, hvor de herefter skriver deres navne og vælger en farve på bil brikkerne.
- Hver spiller starter med en pengebeholdning på 30.000kr.
- Når alle navne er skrevet ind, starter den person der skrev sit navn ind først. Herefter går det på tur, efter den rækkefølge de indtastede deres navne.
- Den givne spiller slå med terningens og rykker det samme antal felter, som terningernes øjnene viser.
- Hvis en spiller rammer et felt, der ikke er købt allerede, som han/hun ønsker at opkøbe, betales beløbet som er angivet på feltet til banken. Feltet omrammes med den samme farve, som ejermandens brikfarve. Købes det givne felt ikke, sættes det på auktion af banken, som giver alle spillere mulighed for at købe det.
- Hvis en spiller rammer et opkøbt felt, betales den givne ejer af feltet et varieret beløb alt efter, om der også befinder sig huse/hotel eller alle af samme gruppe er ejet.
- Ønsker en spiller flere penge, kan han/hun pantsætte sine værdier og få penge af banken for det.
- Rammer en spiller et 'Chance'-felt skal han/hun trække et chancekort, som kan, hvis muligt, gemmes til et senere træk.
- Rammer en spiller et parkeringsfelt, sker der intet.
- Ligeledes gives en spiller en ekstra tur, hvis han/hun slår to ens ved første terningkast.
- Rammer en spiller et 'Go to prison'-felt, rykkes spilleren til 'Prison'-feltet.
- Herefter skal spilleren betale 1000kr til banken i bøde for at kunne komme ud, eller slå to ens eller anvende et chancekort. En spiller gives ikke 4000kr for at passere 'Start'-feltet.
- Hver gang en spiller kommer over 'Start'-feltet gives han/hun et beløb på 4000kr.

[1] Se aktivitetsdiagram for tur, hvori flere af de andre aktivitetsdiagrammer kan indgår, figur 5, s. 14

- Spillet spilles videre og går på tur, indtil alle spillere på nær én er gået bankerot. Den resterende spiller udnævnes som vinder[2].

ID 2: Start/LoadGame
Scope: Matador
Level: User goal
Primary actors: Player1 - Player6.
Stakeholders and interests: Kunden ønsker at der skal kunne tilgås en database, hvorved der vil være mulighed for enten at oprette eller hente et gemt spil. Ved opstart af programmet, åbnes GUI'en og giver valgmuligheden for kunne oprette et nyt spil eller hente et gemt spil.
Preconditions: Spillet startes først, efter spillerne har indtastet relevante informationer ind.
Success guarantee (Postconditions): Spillet startes, når alle relevante informationer er indtastede.
Main Success Scenario (Basic Flows):
1. Bruger åbner applikationen.
2. Applikationen viser knapperne 'Indlæs spil' eller 'Start nyt spil'
3. GUI åbnes med valget mellem nyt spil eller gemt spil.

Extensions (or Alternative Flows):
2. Brugeren vælger 'Nyt spil'.
   a. Brugeren vælger mellem forslag og indtaster påkrævede informationer i følgende orden: Antal spillere (3-6), navn, bilfarve.
2. Brugeren vælger 'Load Game', som gendanner et tidligere gemt spil.
   a. Dropdown-menu vises i GUI'en.
   b. GUI'en viser de gemte spil, hvor datoen for spillet fremgår.
   c. Brugeren vælger et spil, og spillet fortsættes.

ID 3: SaveGame
Scope: Matador
Level: User goal
Primary actors: Player1-Player6.
Stakeholders and interests: Programmet skal kunne gemmes og åbnes igen fra samme udgangspunkt.
Preconditions: Spillet gemmes med alle væsentlige informationer. Spillet må *kun* gemmes når en tur er afsluttes og før en næste begynder.
Success guarantee: Databasen giver brugeren mulighed for at spille fra samme stadie, som spillet blev gemt.
Main Success Scenario (or Basic Flows):
1. En fra Player1-Player6 trykker det røde kryds i venstre hjørne af GUI vinduet.
2. Der udprintes besked af programmet, om spilleren ønsker at gemme spillet.

[2] Se aktivitetsdiagrammer under punkt 3.4

a. Spilleren trykker på 'Ja'-knappen, og databasen opdateres og GUI vinduet lukkes ned.

ID 4: DoAuction:
Scope: Matador
Level: User goal
Primary actors: Player1-Player6.
Stakeholders and interests: Spillerne skal gives mulighed for at købe felter, der ikke er chance-, fængsels eller parkeringsfelter. Ønsker en spiller ikke at købe, skal feltet ryge på auktion og giver alle mulighed for at byde og opkøbe det.
Preconditions: En spiller ønsker ikke at købe et givent felt, og programmet sørger for at samme felt ryger på auktion efterfølgende.
Success guarantee: Programmet sørger for at alle får mulighed for at byde.
Main Success Scenario (or Basic Flows):
1. En fra Player1-Player6 lander på et felt tilgængeligt for køb og køber det.

Extensions (or Alternative Flows):
1. Spilleren vælger ikke at købe grunden og trykker 'Nej'.
   a. Systemet sætter feltet på auktion, og giver de andre spillere mulighed for at købe.
   b. Programmet giver spillerne til at indtaste et vilkårligt bydebeløb i et tekstfield. Bydning går på tur.
   c. Den højstbydende spiller, bliver køberen af feltet og herefter udprinter programmet en meddelelse om at han/hun er blevet ejer af det[3].

ID 5: EndGame
Scope: Matador
Level: User goal
Primary actors: Player1-Player6.
Stakeholders and interests: Programmet skal slutte, når alle på nær én spiller er gået bankerot.
Preconditions: Der findes en vinder af spillet.
Success guarantee: Alle, på nær én spiller, skal gå bankerot, før den sidste spiller kåres som vinder.
Main Success Scenario (or Basic Flows):
1. En fra Player1-Player6 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 Players brik samme antal felter.
3. Spilleren lander på en modspillers opkøbte felt, og skal betale husleje, som han/hun ikke kan.

[3] Se aktivitetsdiagram for auktion

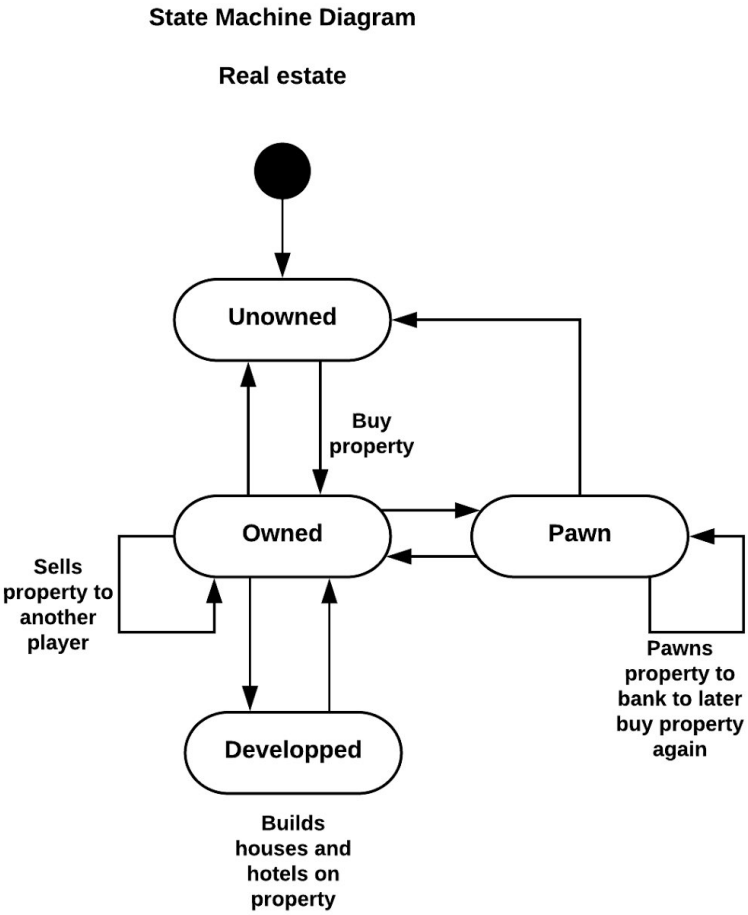
4. Spilleren pantsætter egne grunde, huse og andre værdier, for at betale sin modstander.
5. Spilleren har stadig ikke råd til at betale, og må erklære at han/hun er bankerot.
6. Alle spillere på nær én, går bankerot.

Extensions (or Alternative Flows):

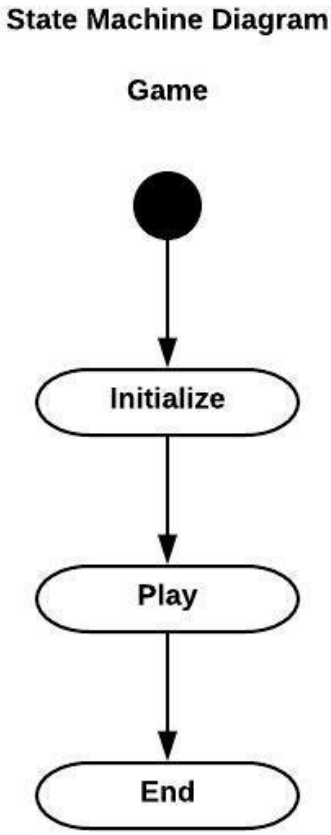
2. Der udprintes besked af programmet, om at spilleren er gået bankerot, og ikke længere er med i spillet.
6. Programmet udprinter besked om, at den sidste spiller er spillets vinder.

3.3 State Machine Diagrams @JM

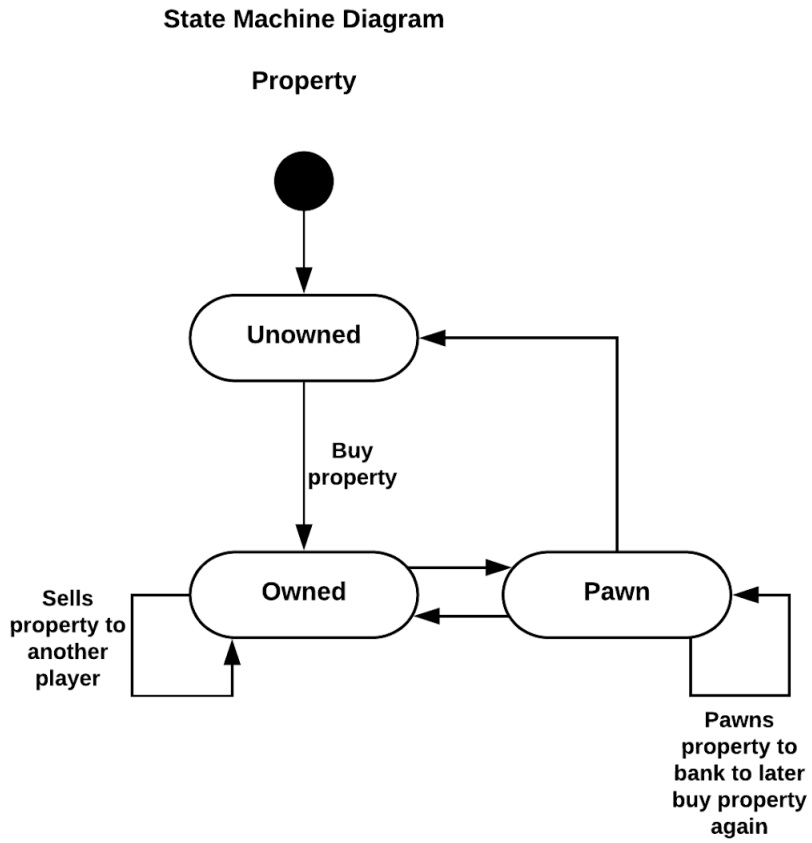
State machines diagrammerne (SMD), giver et overblik over hvilke forskellige stadier og tilstande, en Player (eksempelvis) kan have igennem et spil.
Her symboliserer den sorte udfyldte cirkel startpunktet, hvor pillerne demonstrerer hvordan de forskellige tilstande kan udvikle sig. Her kan der forekomme tekst med beskrivelser af de betingelser for ændring af en tilstand.



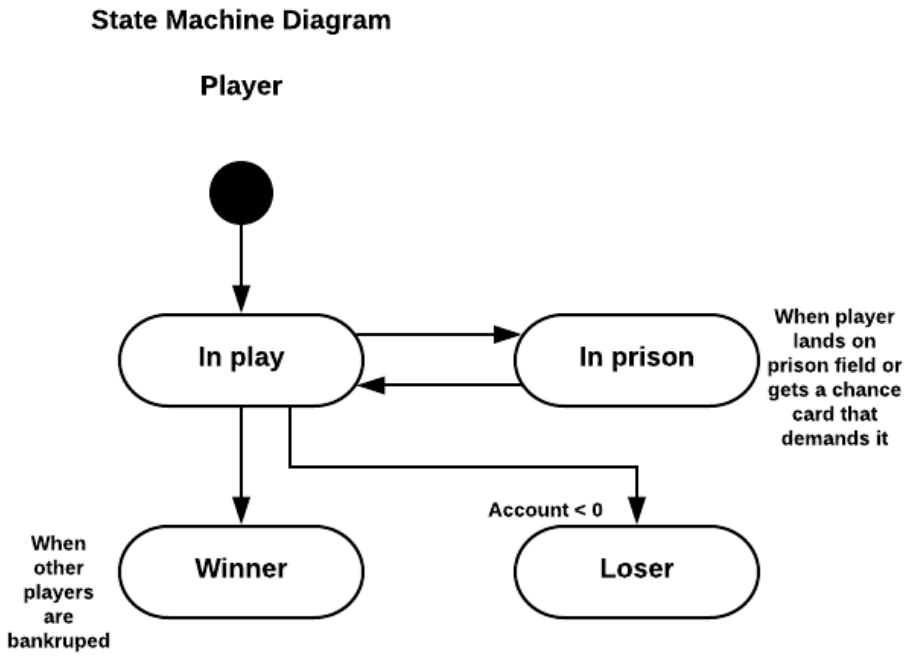
Figur 1: SMD - Real Estate



Figur 2: SMD - Game



Figur 3: SMD - Property



Figur 4: SMD - Property

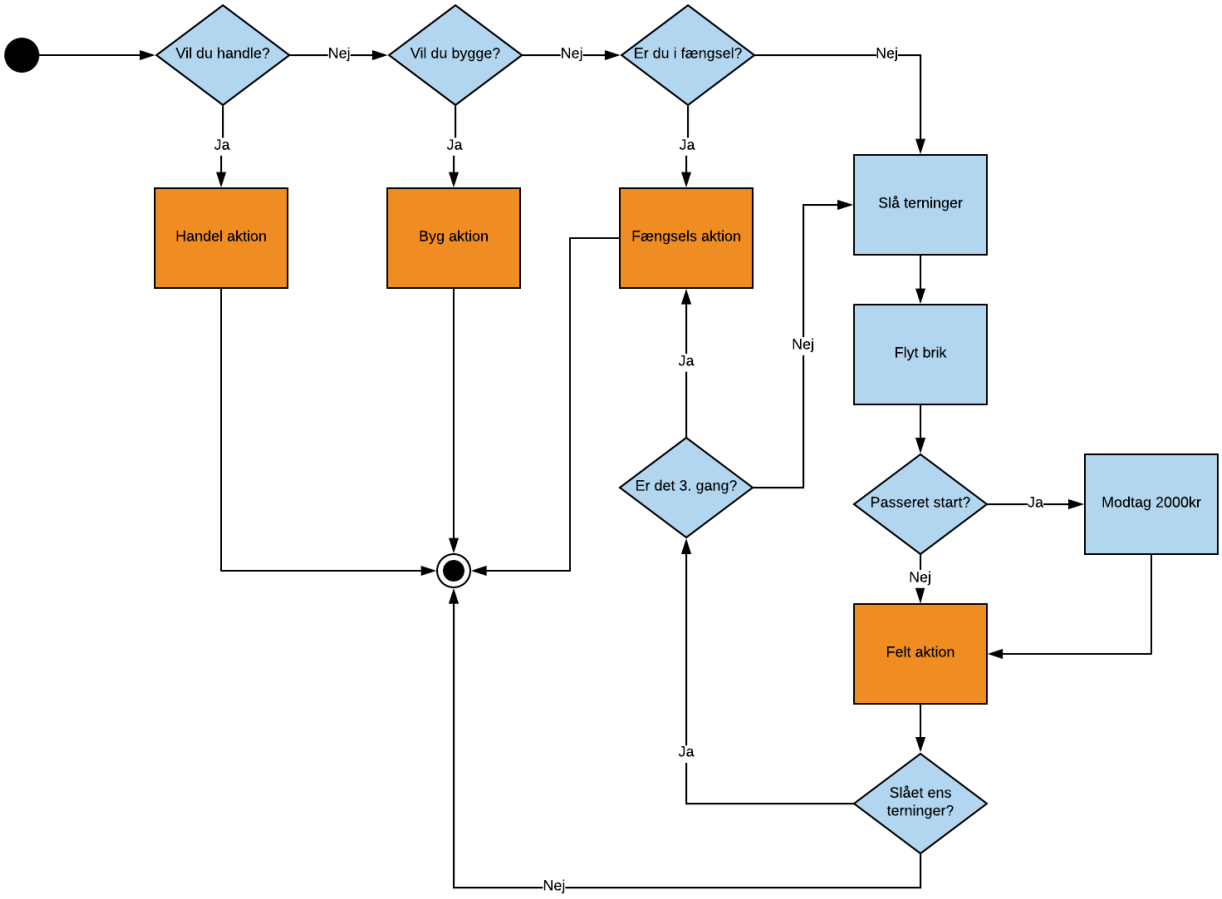
3.4 Aktivitetsdiagrammer @JO @CH

Vores aktiviteter (AD) beskriver her de hovedaktiviteter for nogle af spillets funktioner. Diagrammerne er forklaret via farver:

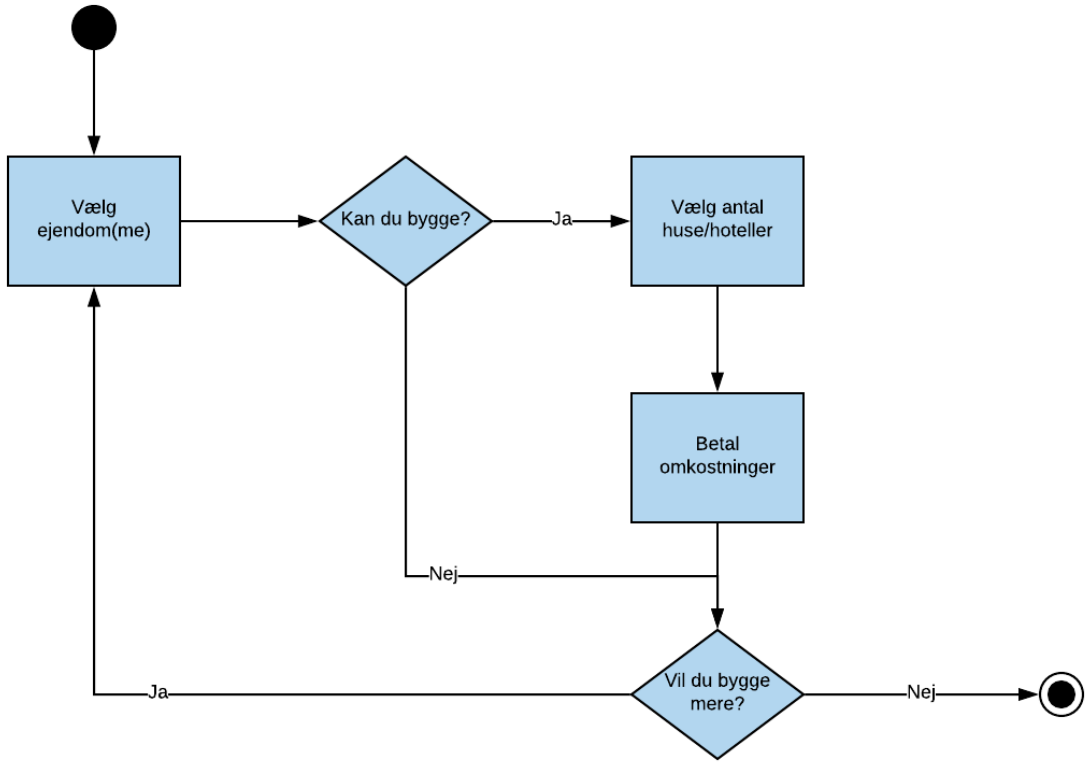
- **Sort cirkel:** Start aktivitet
- **Blå:** Aktivitetens delelementer.
- **Orange:** Beskrives ved andre diagrammer
- **Sort cirkel med cirkel rundt om:** Aktiviteten slutter.

Diagrammerne skal således læses med start fra den sorte cirkel, og i pilenes retning.

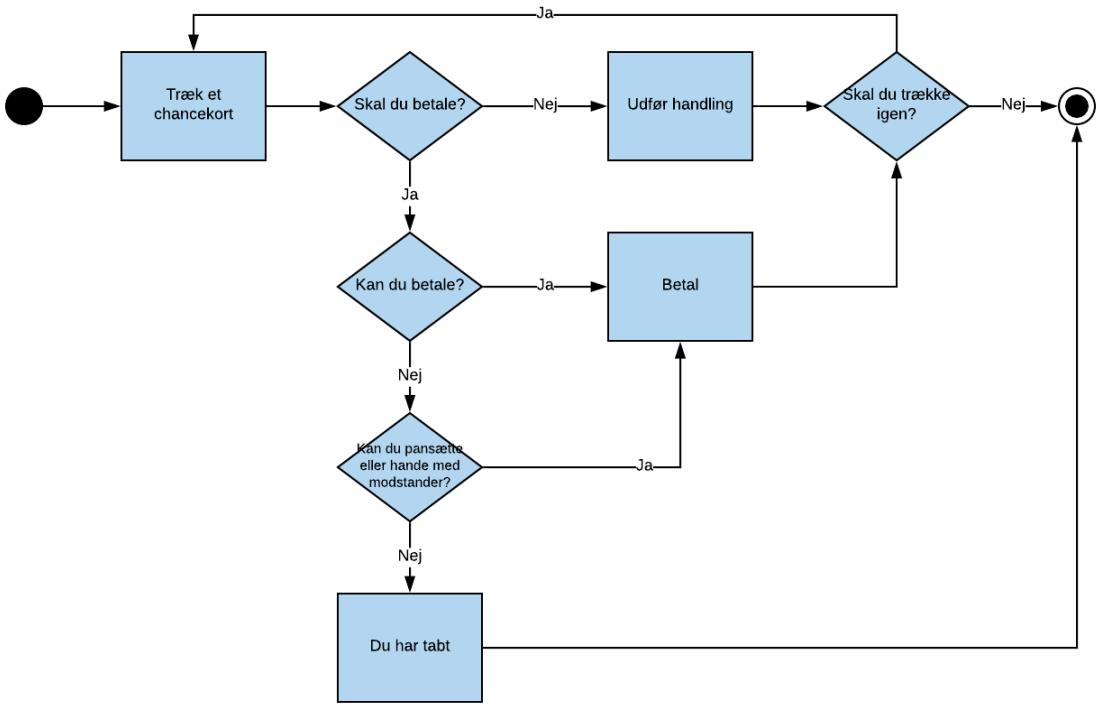
Aktivitet ved spillers tur: Ved en spillers tur kan flere aktiviteter inkluderes, disse er markeret med den orange farve, og er beskrevet i flere af de følgende diagrammer.



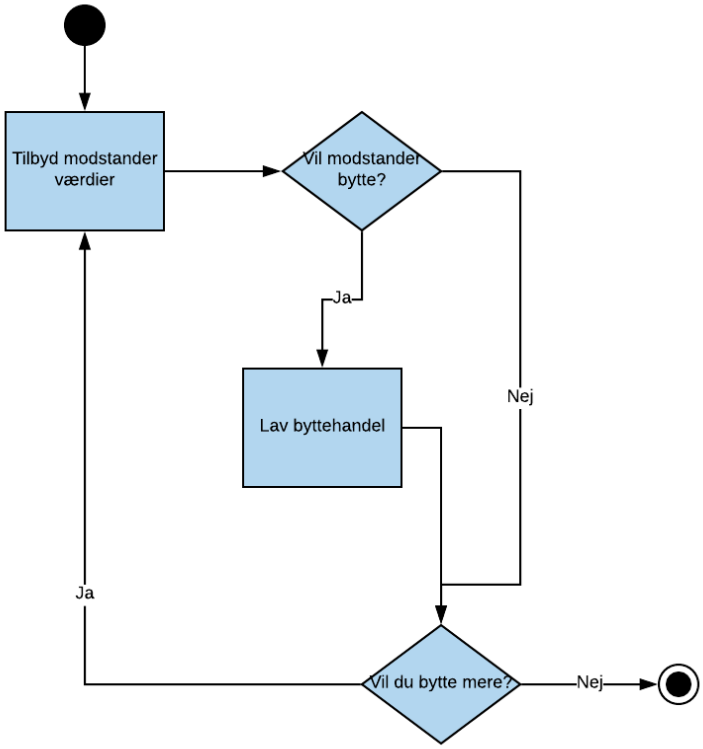
Figur 5: AD - En spillers tur



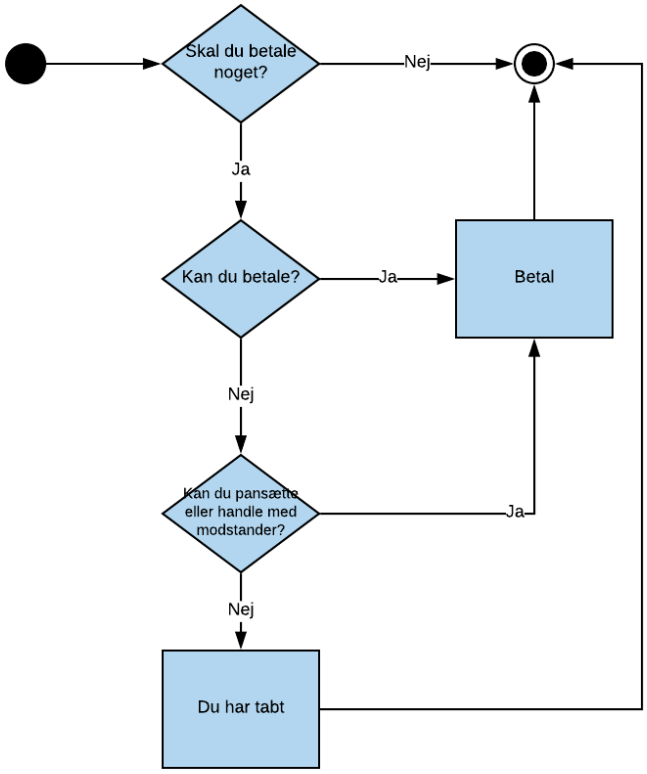
Figur 6: AD - Byggeaktivitet



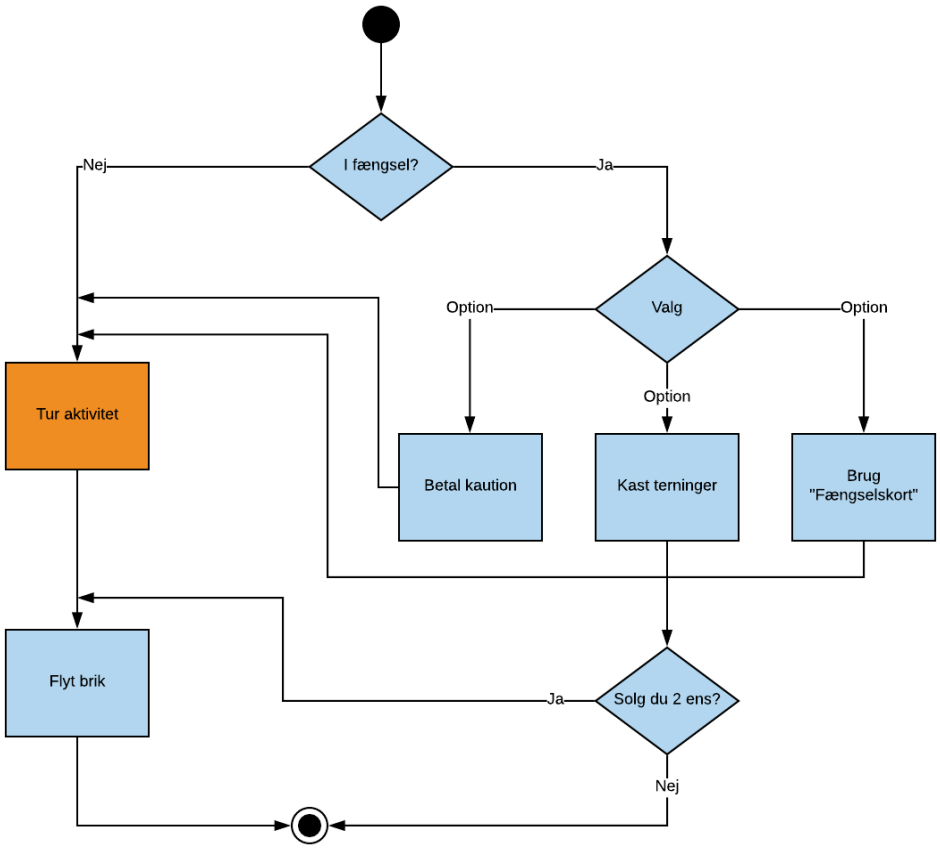
Figur 7: AD - Chancefelt



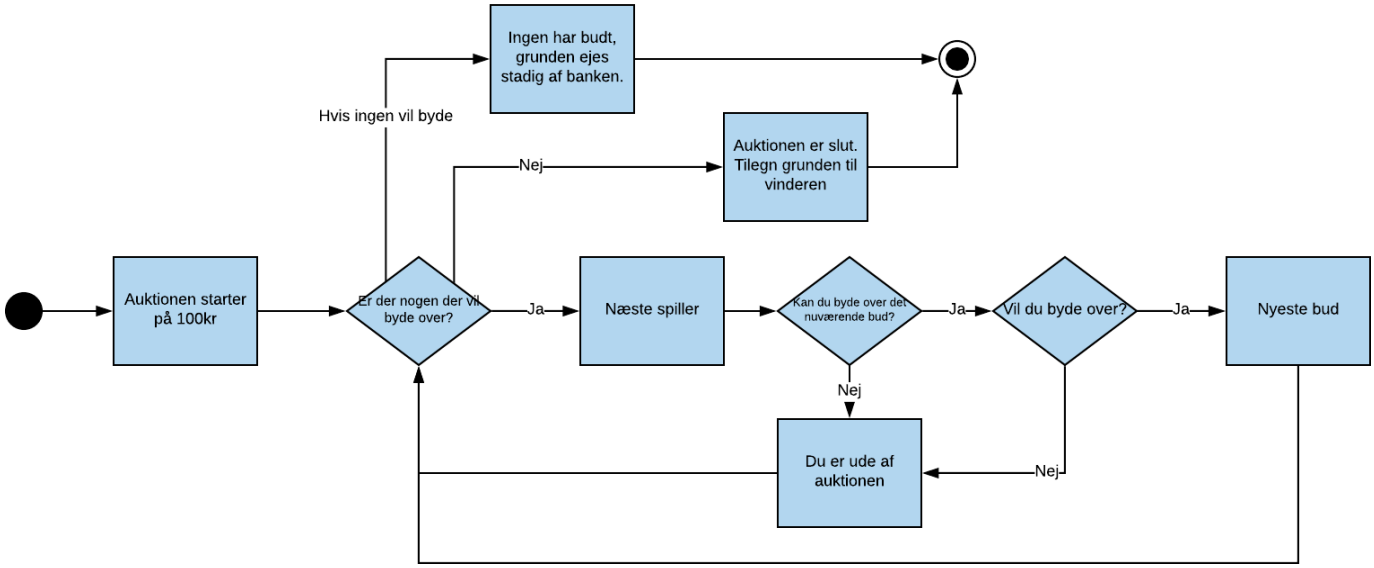
Figur 8: AD - Handleaktivitet



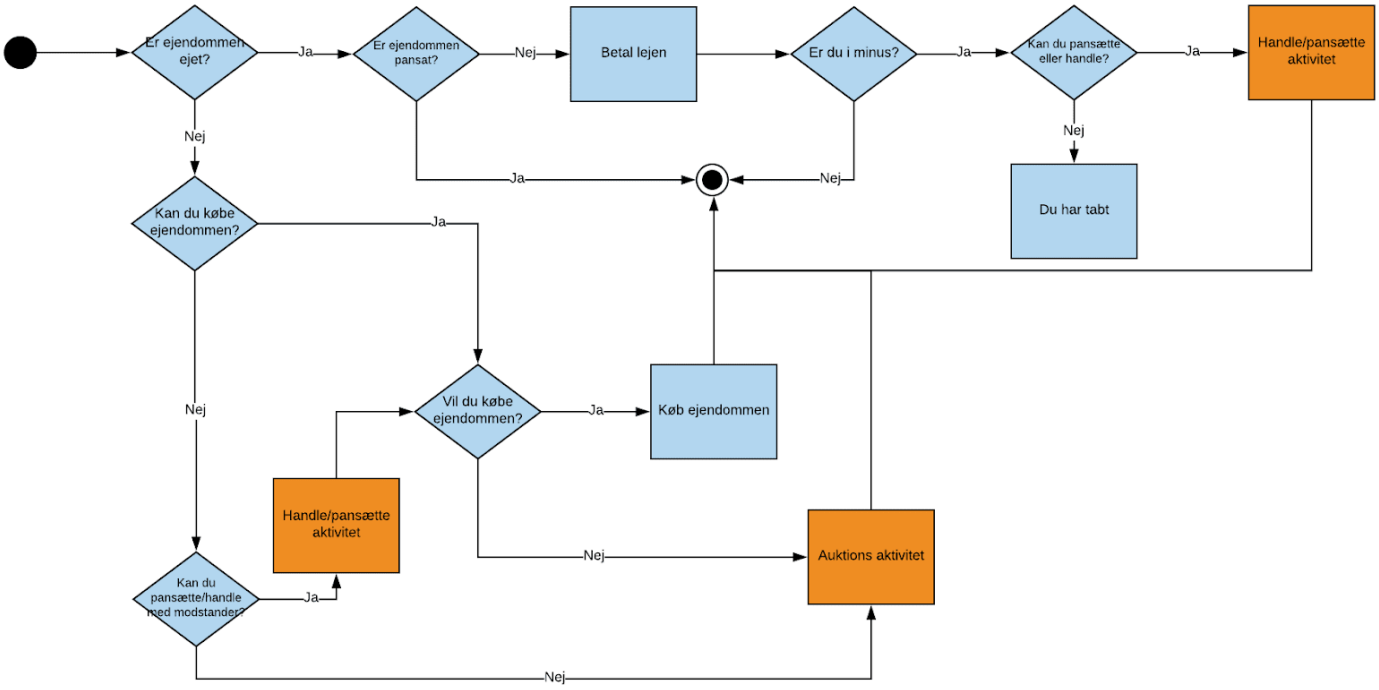
Figur 9: AD - Diverse felter



Figur 10: AD - Fængsling



Figur 11: AD - Auktion



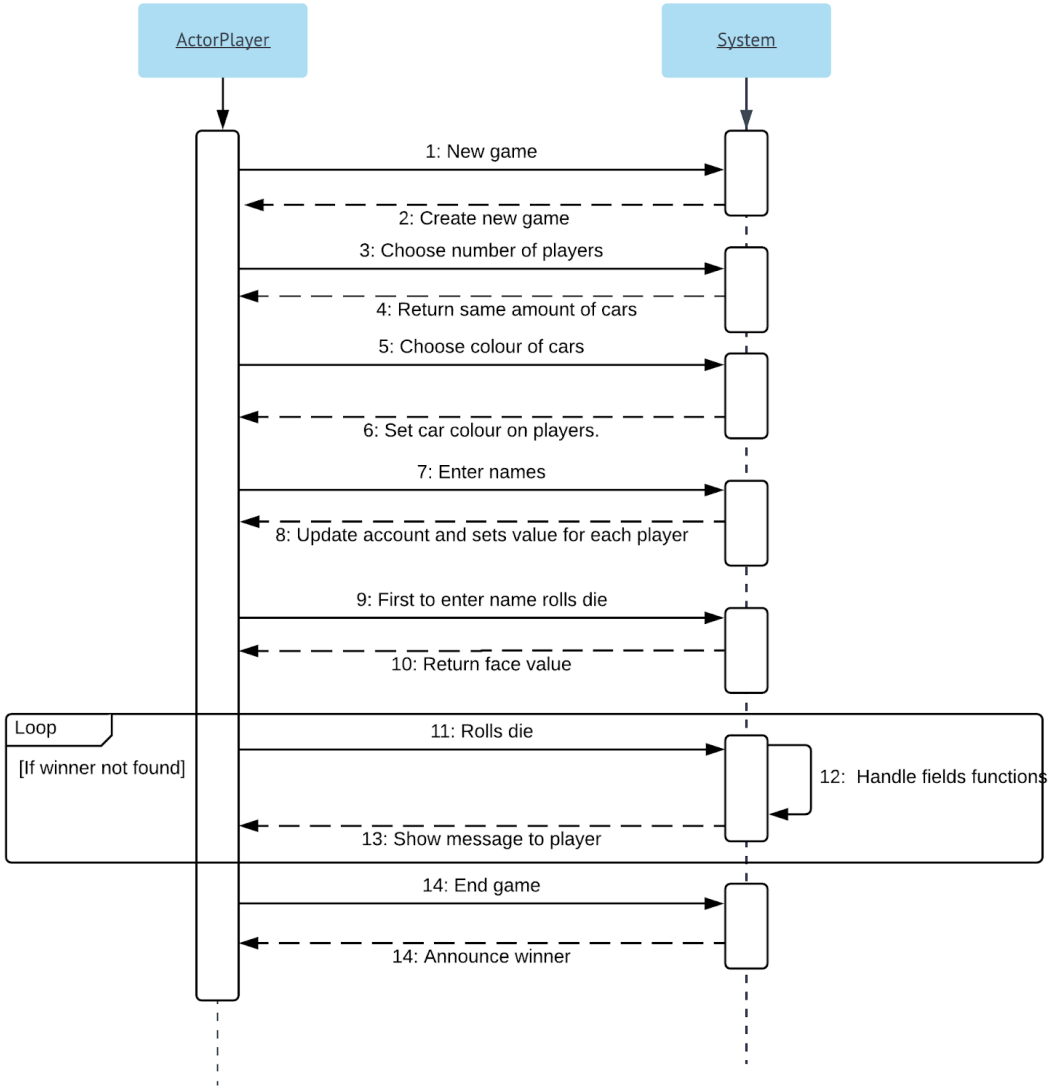
Figur 12: AD - Ejendomsfelt

4. Design (database og software):

4.1 Sekvensdiagrammer @OS

Et sekvensdiagram (SD) er et interaktionsdiagram. Som det fremgår af navnet, viser diagrammet de sekvenser, der foregår mellem aktører og systemet og er behjælpelig til at visualiserer rækkefølgen af handlinger for systemet i sammenhæng med udførelse af bestemte funktionaliteter[4].

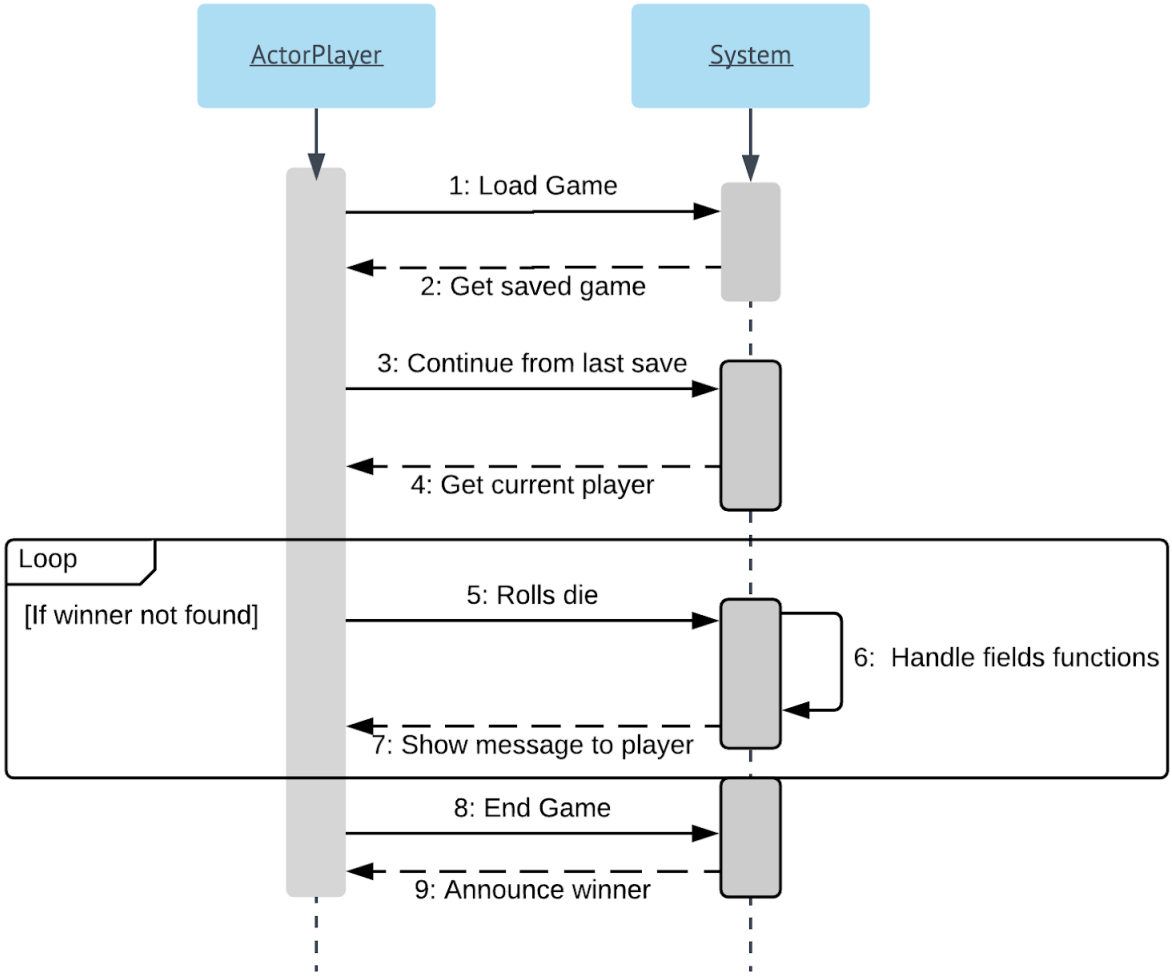
I følgende systemsekvensdiagram vises samspillet mellem en aktør og systemet. I denne gennemgås hvordan et nyt spil startes og hvilke funktioner systemet giver til aktøren. Som nævnt tidligere vælger aktøren nyt spil, og skal derefter indtaste antal spillere, navne, vælge farve. Herefter starter spillet ved at første spiller til at indtaste navn starter, dette fortsætter kontinuert indtil der er én vinder.



Figur 13: SD 'New game'. Figuren inkluderer PlayGame UC og Start/Load Game UC

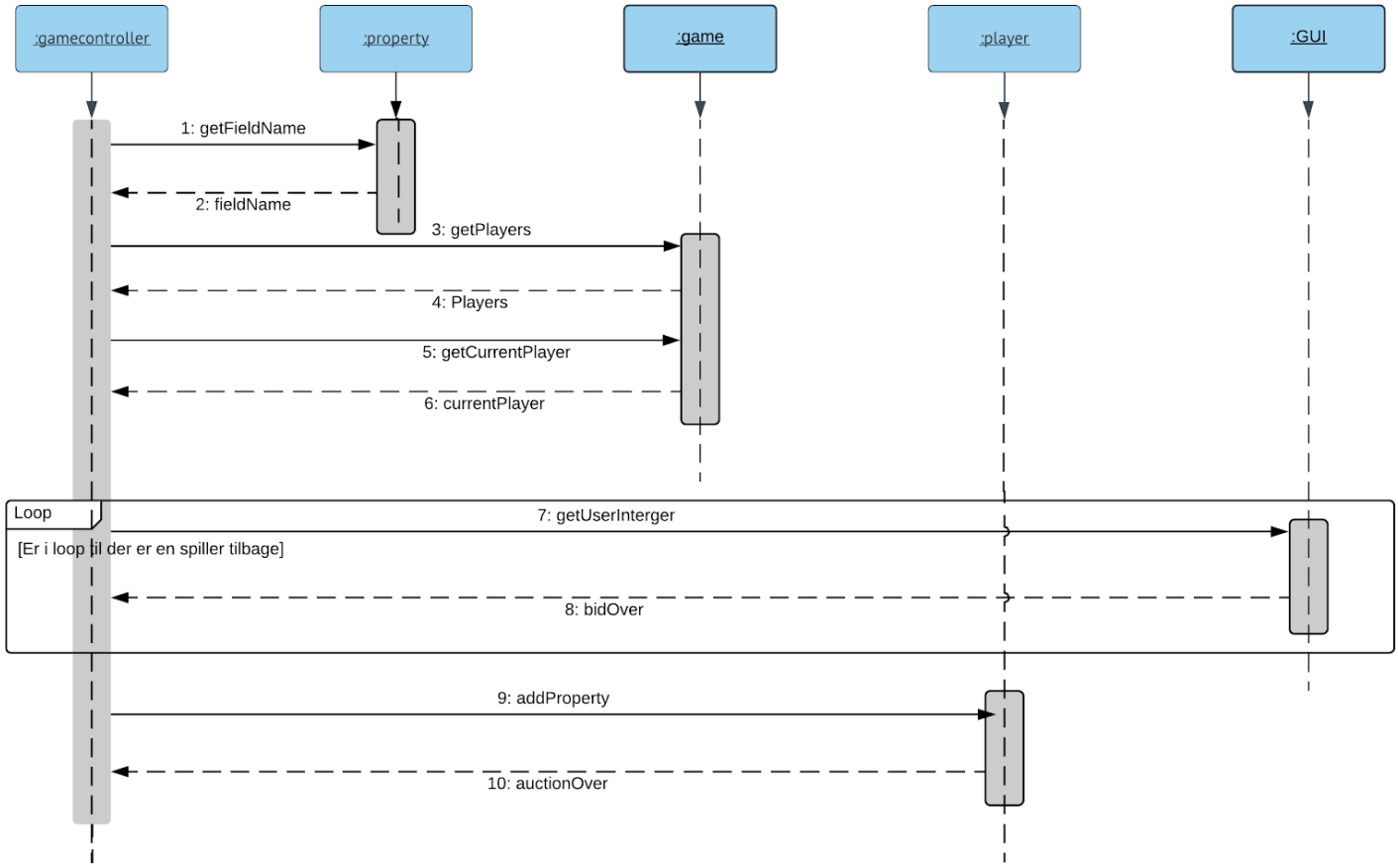
[4] Delvist taget fra slides fra kursus 02315, 2017

Ved dette systemsekvensdiagram vises der hvad der sker når man henter et gemt spil. Når 'Load game' vælges, henter systemet information og fortsætter fra sidst gemte spil. Spillet fortsættes til én spiller er fundet.



Figur 14: SD for 'LoadGame'

Dette er et sekvensdiagram for auction, det viser i hvilken rækkefølge de forskellige information bliver henter fra. Feltet(grunden) der skal til auktion hentes, dernæst hentes spillerne og den nuværende spiller bliver fundet så auktionen kan starte. Auktionen er i et loop til én spiller er blevet fundet, hvorefter grunden bliver tildelt personen der har budt højest.

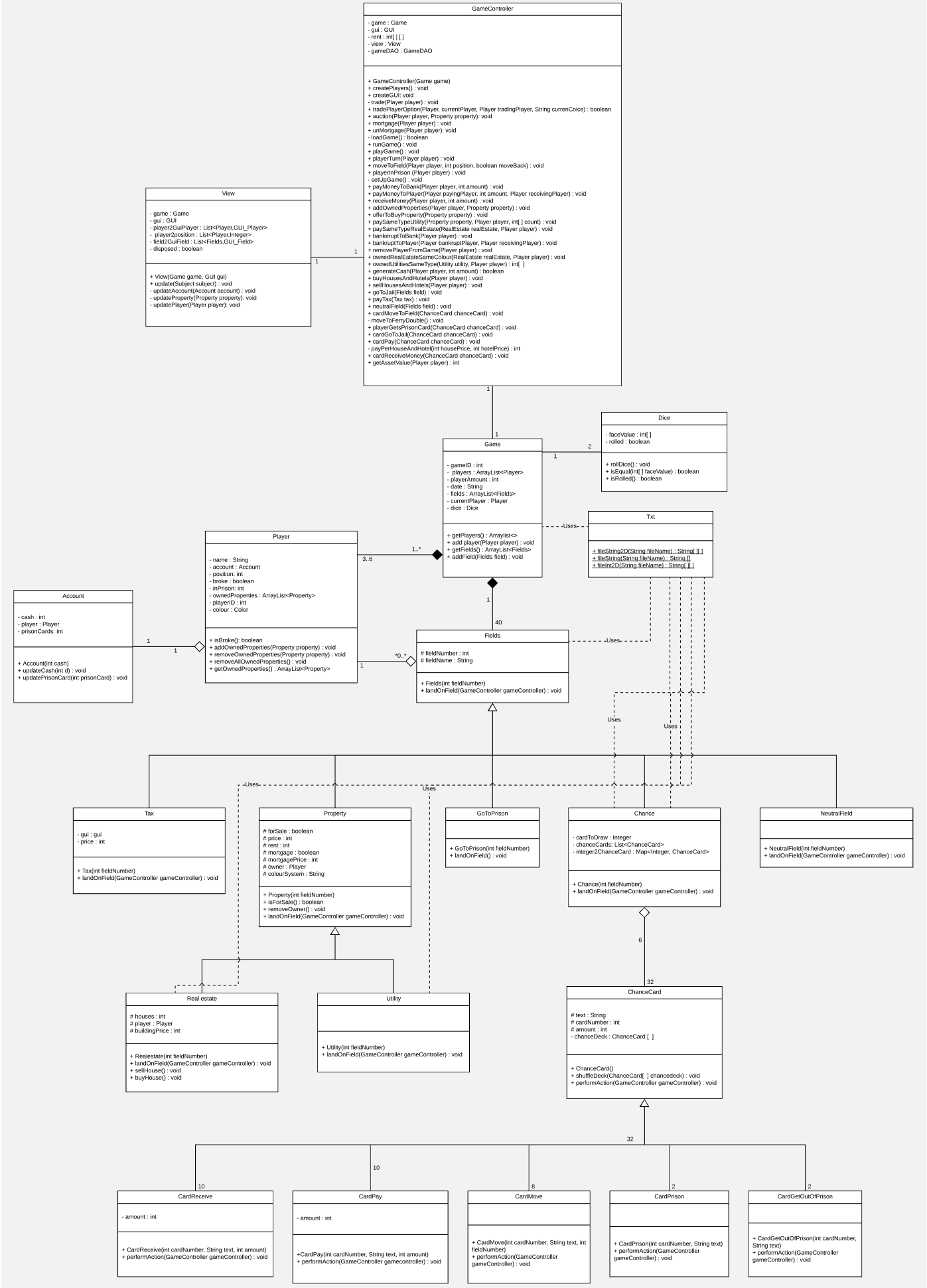


Figur 15: SD for 'DoAuction'

4.2 Klassediagram @CH @OS

Klassediagrammet fungerer som en repræsentation af den statiske visning for Matador applikationen. Diagrammet har til formål at skulle give en oversigt af forskellige aspekter af systemet, men også være behjælpelig til konstruktionen af eksekverbar kode. Diagrammet angiver alle programmets klasser, interfaces og deres attributter og metoder fremgår samt deres associationer imellem. Dette danner ansvar imellem klasserne for såvidt også begrænsningerne for applikationen (se diagrammet på næste side)[5].

[5] Delvist taget fra slides fra kursus 02315, 2017


GameController
- game : Game
- gui : GUI
- rent : int[ ] [ ]
- view : View
- gameDAO : GameDAO
+ GameController(Game game)
+ createPlayers() : void
+ createGUI: void
- trade(Player player) : void
+ tradePlayerOption(Player, currentPlayer, Player tradingPlayer, String currenCoice) : boolean
+ auction(Player player, Property property): void
+ mortgage(Player player) : void
+ unMortgage(Player player): void
- loadGame() : boolean
+ runGame() : void
+ playGame() : void
+ playerTurn(Player player) : void
+ moveToField(Player player, int position, boolean moveBack) : void
+ playerInPrison (Player player) : void
- setUpGame() : void
+ payMoneyToBank(Player player, int amount) : void
+ payMoneyToPlayer(Player payingPlayer, int amount, Player receivingPlayer) : void
+ receiveMoney(Player player, int amount) : void
+ addOwnedProperties(Player player, Property property) : void
+ offerToBuyProperty(Property property) : void
+ paySameTypeUtility(Property property, Player player, int[ ] count) : void
+ paySameTypeRealEstate(RealEstate realEstate, Player player) : void
+ bankeruptToBank(Player player) : void
+ bankruptToPlayer(Player bankruptPlayer, Player receivingPlayer) : void
+ removePlayerFromGame(Player player) : void
+ ownedRealEstateSameColour(RealEstate realEstate, Player player) : void
+ ownedUtilitiesSameType(Utility utility, Player player) : int[ ]
+ generateCash(Player player, int amount) : boolean
+ buyHousesAndHotels(Player player) : void
+ sellHousesAndHotels(Player player) : void
+ goToJail(Fields field) : void
+ payTax(Tax tax) : void
+ neutralField(Fields field) : void
+ cardMoveToField(ChanceCard chanceCard) : void
- moveToFerryDouble() : void
+ playerGetsPrisonCard(ChanceCard chanceCard) : void
+ cardGoToJail(ChanceCard chanceCard) : void
+ cardPay(ChanceCard chanceCard) : void
- payPerHouseAndHotel(int housePrice, int hotelPrice) : int
+ cardReceiveMoney(ChanceCard chanceCard) : void
+ getAssetValue(Player player) : int
View
- game : Game
- gui : GUI
- player2GuiPlayer : List<Player,GUI_Player>
- player2position : List<Player,Integer>
- field2GuiField : List<Fields,GUI_Field>
- disposed : boolean
+ View(Game game, GUI gui)
+ update(Subject subject) : void
- updateAccount(Account account) : void
- updateProperty(Property property): void
- updatePlayer(Player player): void
Dice
- faceValue : int[ ]
- rolled : boolean
+ rollDice() : void
+ isEqual(int[ ] faceValue) : boolean
+ isRolled() : boolean
Game
- gameID : int
- players : ArrayList<Player>
- playerAmount : int
- date : String
- fields : ArrayList<Fields>
- currentPlayer : Player
- dice : Dice
+ getPlayers() : Arraylist<>
+ add player(Player player) : void
+ getFields() : ArrayList<Fields>
+ addField(Fields field) : void
Txt
+ fileString2D(String fileName) : String[ ][ ]
+ fileString(String fileName) : String []
+ fileInt2D(String fileName) : String[ ][ ]
Player
- name : String
- account : Account
- position: int
- broke : boolean
- inPrison: int
- ownedProperties : ArrayList<Property>
- playerID : int
- colour : Color
+ isBroke(): boolean
+ addOwnedProperties(Property property) : void
+ removeOwnedProperties(Property property) : void
+ removeAllOwnedProperties() : void
+ getOwnedProperties() : ArrayList<Property>
Account
- cash : int
- player : Player
- prisonCards: int
+ Account(int cash)
+ updateCash(int d) : void
+ updatePrisonCard(int prisonCard) : void
Fields
fieldNumber : int
fieldName : String
+ Fields(int fieldNumber)
+ landOnField(GameController gameController) : void
Tax
- gui : gui
- price : int
+ Tax(int fieldNumber)
+ landOnField(GameController gameController) : void
Property
forSale : boolean
price : int
rent : int
mortgage : boolean
mortgagePrice : int
owner : Player
colourSystem : String
+ Property(int fieldNumber)
+ isForSale() : boolean
+ removeOwner() : void
+ landOnField(GameController gameController) : void
GoToPrison
+ GoToPrison(int fieldNumber)
+ landOnField() : void
Chance
- cardToDraw : Integer
- chanceCards: List<ChanceCard>
- integer2ChanceCard : Map<Integer, ChanceCard>
+ Chance(int fieldNumber)
+ landOnField(GameController gameController) : void
NeutralField
+ NeutralField(int fieldNumber)
+ landOnField(GameController gameController) : void
Real estate
houses : int
player : Player
buildingPrice : int
+ Realestate(int fieldNumber)
+ landOnField(GameController gameController) : void
+ sellHouse() : void
+ buyHouse() : void
Utility
+ Utility(int fieldNumber)
+ landOnField(GameController gameController) : void
ChanceCard
text : String
cardNumber : int
amount : int
- chanceDeck : ChanceCard [ ]
+ ChanceCard()
+ shuffleDeck(ChanceCard[ ] chancedeck) : void
+ performAction(GameController gameController) : void
CardReceive
- amount : int
+ CardReceive(int cardNumber, String text, int amount)
+ performAction(GameController gameController) : void
CardPay
- amount : int
+CardPay(int cardNumber, String text, int amount)
+ performAction(GameController gamecontroller) : void
CardMove
+ CardMove(int cardNumber, String text, int fieldNumber)
+ performAction(GameController gameController) : void
CardPrison
+ CardPrison(int cardNumber, String text)
+ performAction(GameController gameController) : void
CardGetOutOfPrison
+ CardGetOutOfPrison(int cardNumber, String text)
+ performAction(GameController gameController) : void
Uses

4.3 Database: @GH @JM

Udviklingen af projektets database er foregået i udviklingsprogrammet MySQL Workbench. Databasen indeholder de informationer fra et matadorspil, så som informationer om spillernes kontobeholdninger, ejede ejendomme, position på brættet, det vil sige de informationer, som ændrer sig i løbet af et spil er at finde i databasen. De statiske informationer og værdier ved et matadorspil er ikke at finde i databasen. Priserne på lejen på de forskellige felter, trods de ændres i løbet af et spil grundet bebyggelse af grundene, er ikke at finde i databasen. Disse er udelukket på baggrund af at priserne på lejen for 1,2,3,4 huse og hotel ikke ændrer sig, men er faste, og er derfor skrevet ind i en txt fil og tilgås derfra.
De informationer der gemmes i databasen, er de informationer man ved et virkeligt matadorspil, ville ønske at gemme, hvis spillet skulle pauses, og fortsættes på et senere tidspunkt.

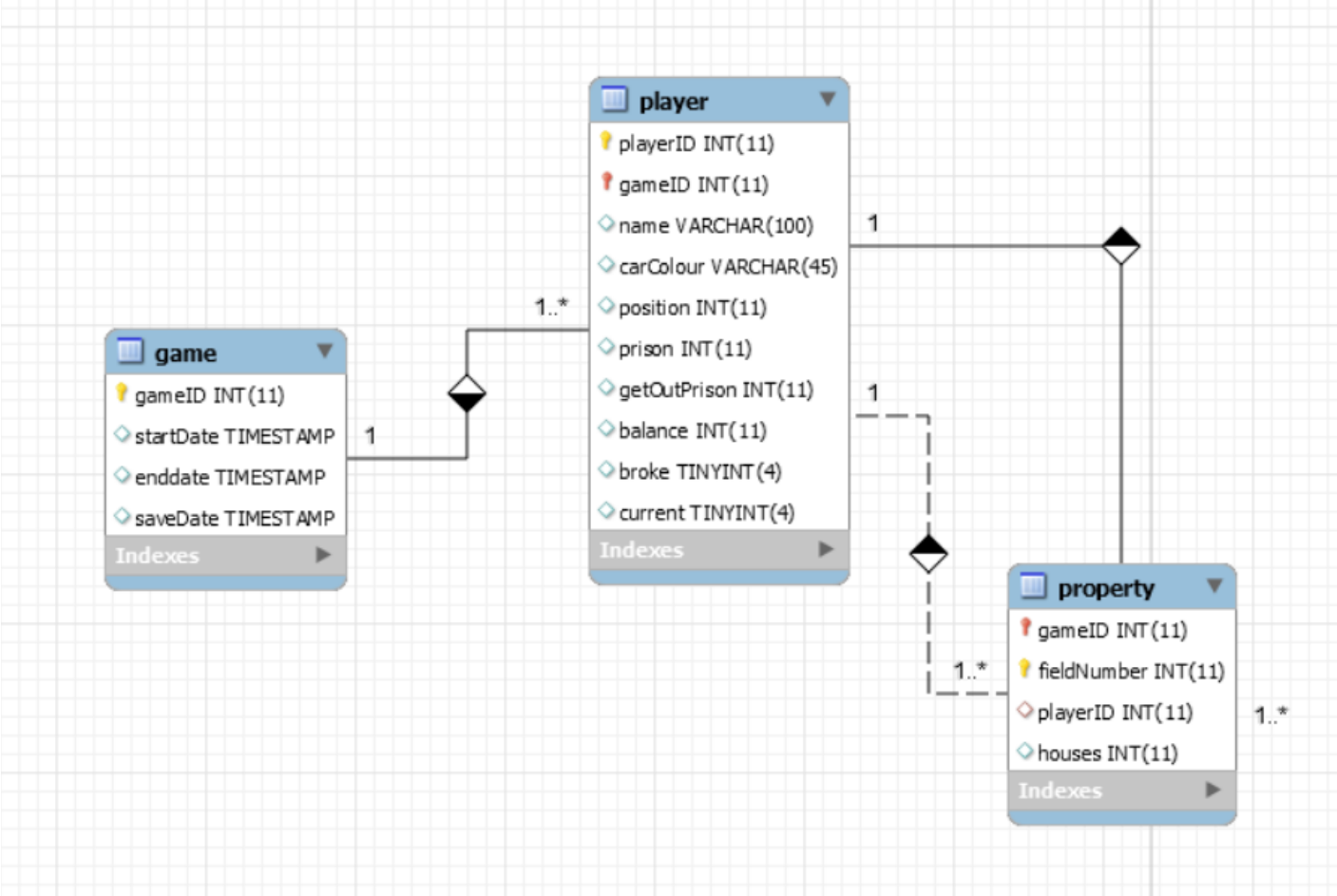
4.4 Databaseskemaer @JO @GH

Til at lave en database, skulle der først og fremmest oprettes databaseskema, som er det logiske design for databasen. Dette giver et overblik over hvor de forskellige instanser (værdier) hører hjemme, når først databasen skal opdateres[6]
Relationerne mellem de forskellige skemaer vises via primary og foreigns keys, som har en endegyldig betydning for hvilke værdier, der lægges i databasen, og hvordan værdier opdatere eller ændres over tid[7]. For at kunne lave de forskellige entities og deres tabeller, hvori programmets værdier skal opbevares, er der via MySQL Workbench først lavet skemaerne og deres tilhørende værdier, som syntes relevante for at et Matadorspil kunne gemmes og deres tilhørende primary og foreign keys. Hertil er der via 'forward engineering' genereret de tabeller, der var behov for.
Forward engineering er den generelle udviklingsproces, når et IT system skal udvikles. Gennem forward engineering, at gå fra en logisk datamodel, i dette tilfælde i form af et skemadiagram, udvikles databasen på den rette baggrund.

Nedenfor ses det udviklede skemadiagram, der ved hjælp af forward engineering dannede baggrund for databasen.

[6] Database System Concepts, s. 42

[7] Database System Concepts, s. 43



Figur 16: Databaseskemaer

4.5Procedures@JM @GH

Stored procedures bruges til at holde Java syntaksen og databasen knyttet, men uafhængigt af hinanden, således at hvis man ønsker andre procedures for ens database, ændres der ikke i selve tabeller og entity for den, men blot hvordan de skal tilgås[8].
Procedures er en måde en udvikler kan oprette egne procedurer og gemme data i database, ved at kalde procedurerne via SQL sætninger i Java. Når man har en stored procedure gives der mulighed for at flere forskellige platforme kan bruge og ændre i databasen. Dette gør det mere anvendeligt, hvis der skal ændres i en procedure, da der kun skal ændres ét sted og ikke for alle platforme, der gør brug af databasen[9].
Til databasen for Matador blev der udviklet diverse procedures med henblik på spil, spillere, ejendomme mf.
Der er udviklet procedure til at oprette spil, spillere og ejendomme i databasen, samt procedures til at læse disse.
Hertil er også procedures oprettet for at opdatere de informationer, der fås i java via at spille spillet, som alle herefter skal gemmes og opdateres i databasen.

[8] Database System Concepts, s. 174

[9] Database System Concepts, s. 173

Nogle af de udarbejdede procedures er oprettet med udgangspunkt i visse Use Cases. Procedures'ne create_game, get_allGames og end_game er alle oprette på baggrund af de tilhørende Use Cases. Som et eksempel på en procedure, se appendix 3.

4.6 Transaction @GH

En transaction er en gruppe SQL kommandoer der har til opgave at hente eller opdatere data i databasen. En transaction afsluttes med en "commit" eller en "rollback", ved commit gemmes transactionen, hvorimod ved en rollback kan en ikke fuldendt transaction afbrydes. Ved en transaction på x antal kommandoer, vil der ved et rollback ske det at, hvis én eller flere af kommandoerne skulle crashe, vil transactionen ikke gennemføres, men derimod gå tilbage til udgangspunktet forinden transactionen startede.

Transaction bruges under projektet i alle de tilfælde, hvor information fra softwaren skal opdateres i databasen, som for eksempel ved metoderne *createPlayers, createGame* osv.
Her bruges transaction funktionerne, når der skal oprettes forbindelse til databasen, og de forskellige værdier opdateres i databasen. Transactionerne der bruges i projektet bruger de ovenfor nævnte kommandoer *commit* og *rollback,* for at gennemføre kommandoen.

I følgende eksempel bruges transaction princippet ved oprettelsen af et spil, der gemmes i databasen.
De grønne fremhævede linjer er de bruge transaction principper.
Der testes i de grønne linjer om den givne information, der kommer fra den resterende del af koden, om den ender i databasen. Hvis databasen opdateres, benyttes "rollback" funktionen ikke, og informationen comittes til databasen.

```
@Override
	public void createGame(Game game) throws SQLException {
		Connection con = connect.getConnection();

		try {
			con.setAutoCommit(false);

			CallableStatement stmt =null;
			stmt = (CallableStatement) con.prepareCall("{call
create_game(?)}");
			stmt.registerOutParameter(1, Types.INTEGER);
			stmt.execute();
			game.setGameID(stmt.getInt(1));
			createPlayers(game, con);
			createProperties(game, con);

			con.commit();

		} catch (SQLException e) {
			// TODO error handling
			e.printStackTrace();
			System.err.println("Some DB error");

			try {
				con.rollback();
				con.setAutoCommit(true);
			} catch (SQLException e1) {
				// TODO error handling
				e1.printStackTrace();
			}

		}
	}
```

4.7 Views og joins @JO

Views kan især anvendes for den data, man ønsker skal være restriktivt, således at alle ikke kan tilgås alle kolonner i en database.
Videre kan views også anvendes som en "virtuel relation", som sker på baggrund af en forespørgelse i SQL, der herefter giver en visuelt overblik over de tabeller, eller dele af tabeller, man ønsker at visualisere for en given person, uden at viewet gemmes og lægges op i databasen[10].
Under udviklingen af databasen er det ikke fundet relevant at gøre brug af restricted views/security view for spillet, da der i det normale Matador ikke holdes noget skjult mellem spillerne. Ligeledes er spillet også beregnet til at foregå på samme computer, hvilket gjorde restriktion af views svært at effektivisere.
Til databasen er der også udviklet flere views. Følgende views er udviklet: boughtproperties, currentbalance, currentgame og currentplayer.
Her er currentgame et forsøg på at lave et view til use casen, Start/LoadGame og SaveGame. Samt boughtproperties, som til dels indeholder uddrag fra flere use cases, såsom DoAuction og LandsOnProperty. I nogle af disse views anvendes 'join on'-funktionen.
Via 'on'-funktion gives der mulighed for at putte et overordnet prædikat på den relation som joines. Ergo fletter join funktionen to tabellers sammen som er fælles for dem.
Dette view kan både gemmes som tabel eller bruges som det er.
Desværre blev det ikke nået at implementere.

Som et eksempel på et view, med brug af Join, se appendix 4.

4.8 Entity-relationship diagram @CH

Databasens tilhørende entity-relationship data model (ER-diagram), giver et overblik over betydninger og samspil mellem databasens tabeller, her gengivet via et konceptuelt skema, der repræsenterer den samlede logiske struktur af databasen.
I ER-diagrammet indgår der entities som Game, Player og Property, samt deres relationer imellem hinanden, samt diverse attributter hvori nogle af værdierne for Matador skal opbevares[11].
I ER-diagrammet er der brugt både primary og foreign keys. Primary keys er i en tabel i databasen, den eller de kolonner der gør alle rækker i tabellen unik. En primary key gør det muligt at finde én bestemt række i en tabel, uden at skulle scanne hele tabellen først. En foreign key er derimod en nøgle, der er til for at forbinde to tabeller, som har en relation. Hvis en foreign key er defineret i tabel 1, refererer den til en primary key i tabel 2. I det udviklede program er foreign keys brugt, blandt andet mellem tabellerne *player* og *property*.

---

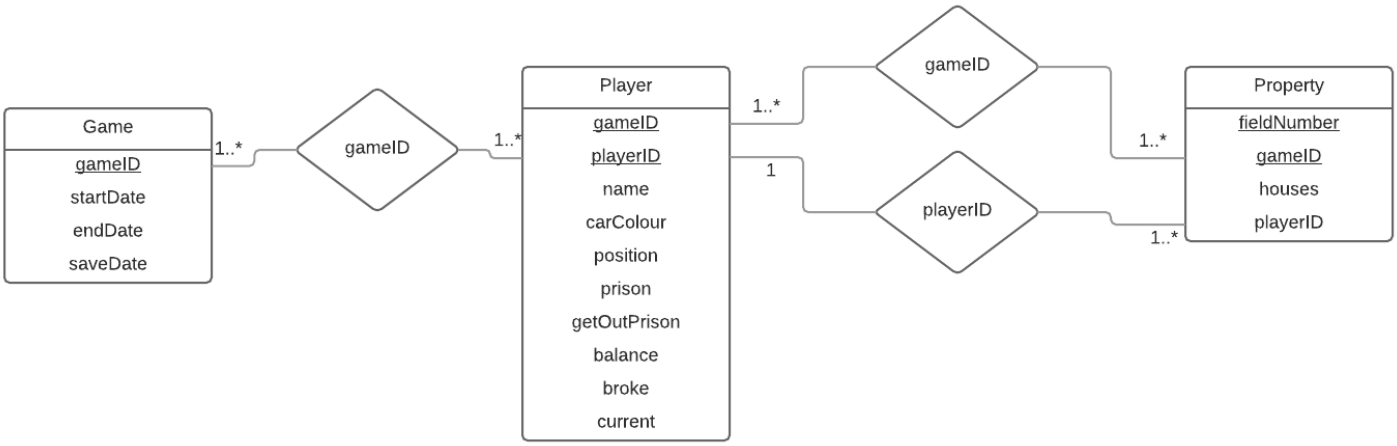
[10] Database System Concepts, s. 121

[11] Database System Concepts, s. 262-263

ER-diagrammet er desuden bygget op ud fra koncepter om normalisering. Normalisering af databasen minimerer redundante data, altså det at den samme data gemmes flere steder. Via normalisering undgås det, at hvis noget skal ændres i en række, behøves det ikke at ændres i andre. Normalisering foregår i dette projekt i op til 3 former. I forbindelse med 3. normal form sørges der for at en attribut ikke afhænger af andet end en primær nøgle[12].

4.9 Overvejelser ift ER-diagrammet:

Da projektet og udviklingen af Matadorspillet hovedsageligt er use case-drevet, var det vigtigt at der i ER-diagrammet indgik et gameID, startDate, saveDate og endDate. Det var væsentligt at en bruger fik muligheden for at deltage i flere spil af gangen, men samtidigt også at kunne gemme undervejs, hvorfor tidspunktet fra start og slut også var relevant. Her måtte der forholdes til, hvad en spiller egentlig ønsker at beholde af informationer og værdier, når han/hun vil lukke et spil og starte op fra samme punkt.
Dette ledte frem til at ens playerID, var væsentligt, da man som selvfølge ønsker at kunne huskes, for at indgå i et spil. Videre også ens navn, bilbrik, position og fra sine modstandere med et ønske om at vinde, var det væsentlig at gemme informationen om andre sad i fængsel eller var broke.



Figur 17: SD for ER-diagram

[12] Database System Concepts, s. 45

4.10 Opkobling af databasen til Java @GH

Måden hvorpå databasen er koblet op til Java sker ved at der oprettes nogle metoder i klassen GameDAO, som skaber en MySQL driver i en classpath, hvortil der skaber en connection mellem databasen og Java. Denne connector-fil hentes fra nettet og tilføres i Java via JDBC'en, som gemmes i mappen 'Referenced Libraries'.
Som det første laves der en public class Connector.
Videre sættes der et 'try'-statement op i forsøg på at koble op til databasen.
I 'try'-statement tildeles forskellige variabler for, hvordan databasen klargøres til opkobling (se eksemplet nedenunder).
Dette involverer bl.a. en String url for en til hvor databasen ligger (jdbc.mysql:// localhost.3306.matador), hvor matador er navnet på selve databasen. Efter skal det etablerer tilkoblingen ved at bruge den information der er lavet i GameDAO-klassen;
Connection = DriverManager.getConnection(url, USERNAME, PASSWORD);
Hertil indføres et andet return statement for connection.
I tilfældet af at 'try'-statementet ikke køres igennem, laves der også et 'catch'-statement, som håndterer, hvis nogen fejl skulle opstå ved opkoblingen af databasen; catch (SQLException e).

```
public class Connector {
    private final String HOST     = "localhost";
    private final int    PORT     = 3306;
    private final String DATABASE = “Matador";
    private final String USERNAME = "root";
    private final String PASSWORD = "";
    private Connection connection;


     public Connector() {

    }

    public Connection getConnection() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            String url = "jdbc:mysql://" + HOST + ":" +
PORT + "/" + DATABASE;
            connection =
DriverManager.getConnection(url, USERNAME, PASSWORD);
            return connection;
        } catch (ClassNotFoundException | SQLException
e) {
            e.printStackTrace();
            System.exit(1);
        }
        return connection;
    }
```

5. Implementering

Ved implementeringen af softwaren er der gjort brug af Model-View-Controller princippet ved at dele koden op i pakker. Ved at dele koden op i pakker efter MVC princippet, bliver koden mere overskuelig og nemmere at vedligeholde i fremtiden.

Som udgangspunkt blev der valgt at skrive koden for ny, men igennem hele processen er der dog hentet inspiration fra tidligere Matador CDIO-opgaver og fra Ekkarts kode, for at validere kvaliteten.
Alle "hjælperklasserne", som i sig selv ikke gør andet end at holde på informationer og metoder, ligger alle i modellaget, som kaldes, når der oprettes et spil i 'Game' i vores controllag. I view laget, ligger det der vises til brugeren. I det udviklede program er viewlaget, board-pakken. Her ligger GUI'en med alle felterne. Controlleren er den del af softwaren der binder view og model lagene sammen, det er her spil logikken ligger, og her alle informationerne samles og sættes sammen til det egentlige spil.

Udviklingen af softwaren er så vidt muligt gjort imens princippet om lav kobling/høj sammenhørighed samt de resterende GRASP principper overholdes.
Lav kobling sikrer at der er så lidt afhængighed mellem klasser, som det er overhovedet er muligt, således at man undgår for så vidt muligt påvirkninger fra andre komponenter ikke opstår ved ændring i en klasse. Høj sammenhørighed hjælper til med at holde den lave kobling, ved blandt andet at fordele ansvar mellem klasserne, så der ikke findes et for stort ansvar i klasserne hver for sig.
Igennem meget af spillets opbygning i modellaget er der gjort brug af nedarvning (extends) af klasser indenfor den samme type, såsom ChanceCard, som er moderklassen til CardMove, CardReceive, CardPay, CardPrison og CardGetOutOfPrison.

Ligeledes var det nødvendigt at gøre for 'Field'-klassen, som er moderklassen til alle typefelter såsom Property, NeutralField, GoToPrison, Chance, RealEstate, Tax og Utility, der dog alle har intergeren 'fieldNumber' og her til en landOnField()-metode, der tager GameControlleren som input tilfælles, men herudover indebærer mange forskellige funktioner og metode hver for sig.
Gennem nedarvning dannes der specialiserede klasser gennem vores superklasse, og giver softwarearkitekturen et bedre og mere overskueligt design, minimerer redundans, der videre også bliver nemmere at teste for.

For at sikre at der tjekkes for om en Player ejer flere felter af samme typer, er der blevet gjort brug af et colourSystem som String. De forskellige farver for felterne oprettes herefter i controllaget i Matador, ved createGame() hvor der senere adderes farver for hvert felt.
Hertil tjekkes der for om samme spiller ejer dem i GameControlleren via metoden; public void ownedRealEstateSameColour, der tager RealEsate og Player som input.

Ved at inddrage filer, gøres koden mere overskuelig og fejl undgås ift. æ, ø og å. Derfor blev klassen Txt oprettet, som henter filerne via datatypen ArrayList, hvorved at beskeder til spillet, priser, chancekort meddelelser, m.m. kaldes via index pladsen for arrayet.
I denne klasse gøres der også brug af en 'try', 'catch' funktion, der gør at der kastes en IOException med beskeden 'error', hvis index ikke findes for beskeden.
Hver gang der indgår beskeder, som skal hentes fra en tekstfil, oprettes der et nyt array, med filnavnet som input. Til flere af tekstfilerne er der blevet gjort brug af array matrix, for at kunne holde orden på eksempelvis flere priser for leje, fremfor at have flere og eventuelt for lange enkelt arrays.
Det var ønsket at bruge Txt-klassen og dens filer konsekvent for GameMessages, men da det gav et bedre overblik at skrive beskederne ind direkte via som en String, da gennemløbning af spillet var det mere overskuelig, da teksten kunne ses klart i syntaksen.

I softwaren bliver der ligeledes brugt HashMap.
For at holde funktioner for Player objektet konsistent med opdateringen af vores view (GUI'en), er der blevet gjort brug af HashMap.
I viewet oprettes en GUI-player, som "linkes" sammen med model-player'en via HashMap og funktionen notifyChanges() gennem hele modellaget, således at gui-objektet og model-objekt, hvad end om det er spillere eller felter, m.m. matches og holdes opdateret parallelt.

Dette er videre relevant for programmets designpattern, som involverer en Observer og Subject-klasse. Observeren sørger for at holde styr på alle notifyChange() betingelser, ved at løbe dem alle igennem og videre de opdaterede værdier videre til Subject-klassen, der sætter dem op i viewet - nu med et opdateret udgangspunkt.

I mange situationer anvendes cast-funktionen for at kunne tildele en typefelt, som en anden typefelt, når der ønskes at anvende nogle af funktionerne, uden selv at skulle tilføje nye metoder til dens oprindelige klasse. Dette er især relevant, når man gerne vil redefinerer datatyper til en anden slags. Det sker eksempelvis via Field-klassen, hvor der castes gennem subklasserne for denne klasse, for at hente relevante metoder og informationer.

På grund af GUI'en har der været en del begrænsninger med at implementerer i hele forløbet. Både visuelt men også funktionelt. Dette skyldes specielt, da GUI'en ikke har været i stand til at tage imod ArrayLists, og derfor har det været nødvendigt at konverterer alle disse om til Strings.
Rent visuelt lemper GUI'en også på brugervenligheden og oplevelsen, da beskeder, der kommer frem ved chancefelterne ikke forsvinder af sig selv, men først når næste spiller lander på et andet chancefelt.
Havde det været muligt, indenfor de besiddende kompetence, ville det havde været oplagt at ændre GUI'en således at en spiller kunne se et chancekort, hvis en given spiller skulle trække et, der kunne gemmes til senere brug. Videre havde det også gjort noget for oplevelsen, hvis der havde et raflebæger i GUI'en til at holde de to terninger.

Havde der været mere tid, var der blevet implementeret regulære udtryk, således at en spiller ikke var i stand til at hedde noget med tegn og tal.

Der har været mange metoder, som indgår i spillet og for nogle har de været svære at implementere, hvorfor det til tider har været svært at overholde "den gode stil" af syntaks, da nogle genveje har måtte tages.

6. Udviklingsproces, udviklingsværktøjer og test

6.1 Udviklingsværktøjer @OS @CH

Eclipse Jee Oxygen
Eclipse er det udviklingsværktøj (IDE) der igennem projektet er brugt til udvikling, af softwaredelen af projektet. Koden der skrives i Eclipse, er skrevet i Java.
Eclipse Jee Oxygen er brugt sammen med Github, ved hjælp af et plug-in, der gør det muligt for de to at arbejde sammen.

Github
Det web-baserede versionsstyringsværktøj Github er ved udarbejdelsen af dette projekt, brugt i sammenhæng med udviklingen af softwaredelen af projektet. Github tillader samtlige af gruppens medlemmer at tilgå det samme softwareprojekt, arbejde på samme tid, gemme egne rettelser og hente andres rettelser og tilføjelser.

MySQL Workbench
MySQL Workbench er et database udviklingsværktøj udviklet til oprettelse, vedligeholdelse mm. af databaser.
Programmet er under udviklingen af projektet brugt under udviklingen af projektets database. Tabeller, Views og Procedures er alle lavet ved brug af MySQL Workbench.

6.2 Udviklingsproces @JM

Under projektet er der blevet gjort brug af udviklingsprocessen Unified Process.
Ved brugen af Unified Process har arbejdet med projektet været iterativt af korte forløb af gangen, og arbejdet med programmet har været tilrettelagt ud fra en use case drevet tilgang, således at programmets funktioner blev beskrevet ud fra brugerens synsvinkel.
Ud fra unified processens fire faser, har arbejdet været som følger:

- **Inception:**

I processens første fase skabtes et overblik stillede krav fra opgave. En overordnet vision for projektet blev udarbejdet samt projektets kravanalyse. Spillets tilhørende database er for projektet et hovedpunkt, da programmet hverken kunne testes, fungere optimalt eller

hensigtsmæssigt uden denne.Databasen blev forsøgt en væsentlige milepæle for arbejdsprocessen, og så det væsentligt at få databasen, så hurtigt på plads, som muligt, Databasen var også den største risici for dette projekt, da det var et nyt område for os som udviklere at gå i gang med.

- **Elaboration:**

I elaborations fasen igangsattes den basale arkitektur for syntaksen. Her blev der taget en beslutning om at udvikle så meget af koden selv som mulig, men med det forbehold at inspiration fra opgavestilleren og fra tidligere og lignende projekter, kunne tages i brug. Da nogle af klasserne for softwaren og deres tilhørende funktioner var mere krævende og væsentlige end andre, blev en del tid brugt på at diskutere hvordan denne del af koden skulle effektiviseres, da risikoniveauet for disse var højere.
Derfor var det også nødvendigt af flere omgange, at pendulerer frem og tilbage mellem analyse- og designdelen af processen i kortere iterationer, for at sikre at klassernes funktion opfyldte de use cases, der var udarbejdet i kravanalysen.

- **Construction:**

I denne fase blev software arkitekturen flettet sammen, og resten af softwaren blev koblet op til databasen, således at at programmet kunne testes.
Der var dog i denne fase ikke meget kontakt til opgavestilleren her, da vi følte os bevidste om, hvilke områder der var løst, og hvilke der manglede.

- **Transition:**

I UP'ens sidste fase blev de mål der i begyndelsen var defineret af processen igennem MoSCow og use cases for softwaren, nået. Herudover blev programmet testet af virkelige brugere, således programmets funktioner og koncept blev testet, i forhold til om de levede op til den brugervenlighed, der var tilstræbt for softwaren.

6.3 Test @CH @OS

Under udviklingen af software programmer, vil der gennem processen ske fejl, fejl som under udviklingen ikke har været en fejl i udviklernes øjne. For at undgå disse fejl er der til projektets software udarbejdet flere tests, for at finde og rette de væsentlige fejl inden produktet udleveres til kunden. Der er til test af selve koden oprettet flere JUnit tests. Til test af programmet som helhed er der udarbejdet brugertest, udført af flere personer uden kendskab til projektet, for at teste brugervenligheden, forbedre kvaliteten og finde flere fejl[13].

JUnit tests
De udførte JUnit tests er inkluderet i programmet.

[13] Delvist tager fra CDIO3, gruppe 37, kursus 02314, 02315, 02313

De udførte JUnit test Cases kan alle køres fra den oprettede JUnit test Suite "AllTests". Se andre tests under appendix 5.

Test 1: receiveMoneyTest

Test Case ID	TC1
Referat	Automatiseret test af receiveMoney
Krav	Spillerens pengebeholdning bliver opdateret med det rette beløb
Betingelser før	Spillerens penge er 30.000
Betingelser efter	Spillerens penge er opdateret
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	10.000
Forventet resultat	40.000
Faktisk resultat	40.000
Status	Bestået
Testet af	Christian Høj
Dato	05/05-2018
Testmiljø	Eclipse Jee Oxygen på Windows 8

Test 4: moveToFieldTest

Test Case ID	TC4
Referat	Automatiseret test af moveToField
Krav	Spillerens position bliver opdateret til den nye position
Betingelser før	Spillerens position er 4
Betingelser efter	Spillerens position er opdateret
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	ny position=10
Forventet resultat	10
Faktisk resultat	10
Status	Bestået
Testet af	Christian Høj
Dato	05/05-2018
Testmiljø	Eclipse Jee Oxygen på Windows 8

Test 5: payMoneyTest

Test Case ID	TC5
Referat	Automatiseret test af payMoney
Krav	Spillerens pengebeholdning bliver opdateret med det rette beløb
Betingelser før	Spillernes penge er 30.000

Betingelser efter	Spillernes penge er opdateret
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	1000
Forventet resultat	29000 og 31000
Faktisk resultat	29000 og 31000
Status	Bestået
Testet af	Christian Høj
Dato	05/05-2018
Testmiljø	Eclipse Jee Oxygen på Windows 8

Monkey test:

Der er lavet mange monkey tests, for at tjekke om applikationen levede op til visionen, lavet på baggrund af kravsanalysen. Her blev der fundet flere "skønhedsfejl" nogle af større betydning og andre af mindre. Disse fejl er forsøgt rettede bedst muligt.

Brugertest

Følgende brugertests er udført på det færdige spil. Nedenfor er en oversigt over stillede spørgsmål i testen[14].

Spørgsmål	**Svar**
Forstår du hvad spillet går ud på, uden producenternes hjælp?	Håndbogen var nødvendig for at kunne bruge programmet, men selve spillet var simpelt og nemt at bruge.

[14] Delvist tager fra CDIO3, gruppe 37, kursus 02314

Har du brug for en brugervejledning for at spille spillet?	Da man modtager meget information i GUI'en er en brugervejledning ikke meget nødvendig. Det er ligesom med det "normale" Matador, hvor at man selvfølgelig kigger i reglerne fordi, det ikke er alting man kan huske.
Er spillets tekster for at gennemføre spillet forståelige?	De tekster man modtager fra GUI'en er ganske simple og derfor meget forståelige.
Er spillet brugervenligt?	Det menes at spillet næsten ikke kan forsimples mere, i forhold til alle de funktioner spillet har.
Tog det for lang tid at spille spillet færdigt?	Ja, da matador generelt er et langt spil, gav det således mening at tidsforbruget var omtrent identiske fra det fysiske til det virtuelle.
Har du evt. forbedringer til spillet?	Forbedringer til spillet kunne være interface, det eneste skulle være smuksering af de nuværende ting for at få en bedre visuel følelse. Når man modtager chance-kort bliver beskeden stående indtil der kommer ny information, hvilket burde ordnes. Hertil kan det være frustrerende at trykke på så mange knapper igennem spillet.

Brugertesten er udført af personer uden kendskab til projektet eller indmaden af projektet. Testen er udført af brugerne uden interaktion fra projektets deltagere, dvs. test-brugerne har været i kontrol under hele testen, og er blevet observeret af en eller flere af projektets deltagere. Medfølgende noter er skrevet ud fra brugerens tanker, kommentarer og reaktioner ved test af programmet

7. Håndbog

Se medfølgende håndbog for installation og spilleregler i zip-filen.

8. Konklusion

Efter udviklingen af projektet kan der nu konkluderes, at udviklingen af softwaren i projektet har set sig mere kompliceret og krævende, end først antaget. Dette er bl.a. grundet den udleverede GUIs begrænsninger. Ikke desto mindre er den endelige udarbejdede software mere end tilfredsstillende, i forhold til de lokaliserede analytiske krav, der er beskrevet i starten af rapporten. Ud fra den udarbejdede kravliste, er de mest centrale og vigtigste funktioner for et virkeligt Matadorspil at finde i det endelige udviklede spil. Samtidig medfølger håndbog for installation og regelsæt, således at det opnår så stor brugervenlighed som muligt.

Den hertil udarbejde database giver de fornødne og påkrævet muligheder for at kunne gemme og genopstarte samme spil, ved at gemme spillets data i databasen. Der er til databasen både udarbejdet views og procedure, samt oprettet alle de nødvendige tabeller der skal til for at møde de opstillede krav. Ydermere er funktionerne omkring transactions også brugt til at optimere projektets resultat.

Ud fra det samlede udarbejdet materiale kan det derfor konkluderes, at det færdige spil efterlever de vigtigste, både funktionelle og ikke-funktionelle krav, der kræves. Det udarbejdede materiale giver brugeren en virkelig fornemmelse af det rigtige matadorspil, på en virtuel platform.

9. Referencer

Litteratur

- Abraham Silberschatz, *Database System Concepts 6th Edition.*
- *CDIO3, gruppe 15 opgave, 02313, 02314, 02315*
- *Kursusslides 02315, 2017*
- *CDIO3, gruppe 37, kursus 02313, 02314, 02315*

Websider

- http://www2.imm.dtu.dk/courses/02312/Heldag/matador.pdf - Matador regler

- https://www.lucidchart.com - Brugt til udarbejdelse af diverse diagrammer.

- https://www.techopedia.com - Brugt til diverse

10. Appendix

10.1 Appendix 1 (Use Cases)

Fully dressed[15]**:**
Use Case ID 1: PlayGame:
Scope: Matador
Level: User goal
Primary actors: Player1-Player6.
Secondary actor: DTU (Ekkart og Bjarne)
Stakeholders and interests:

- Vores kunde ønsker et Matadorspil, hvor det er et must, at det skal kunne spilles af voksne (3-6 spillere).
- Det er et must at programmet indeholder 1-3 forskelligfarvede bilbrikker samt 36 chancekort og én spilleplade med 40 felter.
- Hver spiller skal gives muligheder for at opkøbe, de felter, som ikke er et chance-, start, fængsels- eller parkeringsfelt. Købes et felt, vil det blive omrammet af den samme farve, som den givne spillers bilbrik har.
- Programmet skal kunne aflæse terningens værdi, og rykke den givne spille, samme antal felter, som terningens øjnene viser. Programmet skal også kunne aflæse, hvis en spiller slår to ens, som vil give ham/hun mulighed for at slå igen.
- Kunden ønsker at systemet kan aflæse terningens værdi og rykke en given spiller samme antal felt som er lig terningens øjnene. Samt aflæse, når de slås to ens, således at en spiller får et ekstra slag.
- Kunden ønsker, at købsfeltet ryger på auktion, og giver andre spillere mulighed for at købe det, hvis den givne spiller ikke selv vil
- Kunden ønsker, at systemet kan aflæse, når en spiller lander på et 'De fængsles'-felt, hvorefter den givne spiller rykkes til fængselsfeltet.
- Kunden ønsker, at 'Parkering'feltet er et frifelt, hvor der intet sker.
- Det ønskes, at systemet kan aflæse, når Player1-Player6 passerer 'Start'-feltet, hvorefter spilleren gives 4000kr.
- Programmet skal kunne læse, når en modstander lander på et opkøbt felt, således at der opkræves et beløb for leje, der varierer.
- Kunden ønsker, at spillet sluttes, når alle undtagen én spiller er gået bankerot, hvorved at den resterende spiller udnævnes som vinder.

Preconditions: Spillet kan først startes, når nødvendige informationer (såsom navn og antal spillere er indtastet i programmet).
Spillet er færdigt, når alle spillere undtagen én er gået bankerot.
Spillerne starter alle med 30.000kr.
Spillet skal kunne gemmes og opstartes fra samme punkt.
Success guarantee (or Postconditions): Spillet spilles indtil en vinder er fundet.

---

[15] Delvist tager fra CDIO3, gruppe 15, kursus 02313, 02314, 02315

Main Success Scenario (or Basic Flows):

1. En fra Player1-Player6 opstarter spillet på computeren.
2. En fra Player1-Player6 trykker på 'Roll'-knappen og systemet respondere ved at vise værdien af terningens øjnene.
3. Systemet aflæser værdien af terningens øjnene og rykker den givne Player's brik samme antal felter.
4. En fra Player1-Player6 lander på et felt tilgængeligt for køb
5. En fra Player1-Player6 lander på et opkøbt felt.
6. En fra Player1-Player6 et 'Parking'-felt.
7. En fra Player1-Player6 lander på et 'Go to prison'-felt.
8. En fra Player1-Player6 lander på et 'På besøg'-felt.
9. En fra Player1-Player6 lander på et 'Chancekort'-felt.
10. En spiller ønsker at bygge på sin grund.
11. Spillerne af spillet, ønsket at gemme spillet og lukke det ned, for senere at spillet det fra samme udgangspunkt.
12. En fra Player1-Player6 har ingen penge.
13. Player1 kommer over 'Start'-feltet.

Player1-Player6 repetierer punkt 1-13 indtil:

1. Player1-Player6 går bankerot
2. Player1-Player6 udnævnes vinder og spillet sluttes.

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.
   1.1. Køber spilleren feltet, trækkes beløbet på grunden fra den givne spillers pengebeholdning.
2. Spilleren vælger ikke at købe grunden og trykker 'Nej'.
   2.1. Systemet sætter feltet på auktion, og giver de andre spillere mulighed for at købe.
   2.2. Den højstbydende spiller køber feltet.
   2.3. Købsbeløbet trækkes fra den givne spillers konto.
   2.4. Det opkøbte felt omrammes med den samme farve, som den givne spillers brik

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

1. Aflæser systemet, hvor mange huse og hoteller, der er på feltet.
   1.1. Systemet meddeler at den givne spiller, opkræves et givent beløb til feltets ejermand. .
      1.1.1. Hvis den givne spiller ikke kan betale, kan han/hun pantsætte egne grunde, huse og hoteller til banken.
      1.1.2. Kan den givne spiller vælge at sælge en eller flere grunde til en modstander.
      1.1.2.1. Betaler modstanderen til sælgeren.
      1.1.2.2. Overføres beløbet til sælgerens konto.
      1.1.2. Banken udbetaler penge til den givne spiller.
   1.2. Den givne spiller betaler et givent beløb til feltets ejer

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

1. Udprintes en meddelelser om, at spilleren at han/hun er 'Safe'.

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

1. Udprinter systemet, at den givne spiller fængsles.
2. Rykkes den givne spiller til 'Prison'-feltet.
   2.1. Den givne spiller gives i denne situation ikke 4000kr, når han/hun passerer 'Start'.
1. Den givne spiller må forsøge at komme ud af fængsel ved efterfølgende slag.
   3.1. Spilleren vælger at betale 1000kr Til banken.
   3.2. Spilleren bruger et gemt chancekort, der kan få ham/hende ud af fængsel.
   3.3. Spilleren får tre terningeslag.
      3.3.1. Spilleren slår to ens, kommer ud af fængslet og rykker samme antal, som terningerne viser.
1. Den givne spiller kommer ud af fængsel.

7a: For hver gang en spiller, lander på et 'På besøg'-felt:

1. Afgives en meddelelse om at spilleren er på besøg i fængslet

8a: For hver gang en spiller lander på et 'Chancekort'-felt:

1. Trækker systemet et tilfældigt kort, og udprinte dets meddelse.
   1.1. Omhandler chancekortet muligheden for at komme ud af fængslet, kan den givne spiller gemme det til senere brug.
1. Aflæser systemet chancekortet funktion og rykker spilleren efter samme princip.

9a: Hver gang en spiller ønsker at bygge på sin grund:

1. Trykker han på en knap 'køb hus', hvorved at der trækkes et varieret beløb fra hans/hendes pengebeholdning, og der sættes et hus på grunden.
2. Ønsker spilleren at købe hotel, kræves det at der først er bygget fire huse på grunden.
   2.1. Der må her trykkes på en knap for at indkøbe hotel, hvorved at der trækkes et varieret beløb fra hans/hendes pengebeholdning.

10a: Hvor gang en spiller ønsker at indtjene penge:

1. Kan han/hun ville at sælge en grund, hus, hotel til banken eller en anden spiller.
   1.1. Sælgeren får beløb sat ind på sin konto og der trækkes samme beløb fra Køberen
   1.2. Banken betaler til sælgeren, og han/hun får beløbet sat ind på sin konto. →
2. Spilleren vælger at pantsætte sin værdier til banken, og banken giver spilleren halvdelen af værdibeløbet i kontanter til spilleren.
   2.1. Sælgeren køber værdierne tilbage for et fuldt beløb.
   2.2. Modstanderne køber de pantsatte værdier for det fulde beløb og får rådigheden over dem.

11a: Hver gang en spiller kommer over 'Start'-feltet:

1. Gives den givne spiller 4000kr til sin konto.

12: Når en spiller går bankerot:

1. Udprintes en meddelelse, om at spillet at personen har tabt.
   1.1. Spillet spilles videre af de andre spillere indtil alle undtagen én er bankerot.

13a: Når alle spillere undtagen én er gået bankerot:

1. Meddeles den sidste spiller, at han/hun har vundet.

Special Requirements:

- Der er 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.
- Spillet skal kunne lukkes ned og gemmes, således at spillet starter op med samme udgangspunkt næste gang, det åbnes.
- Spillet skal gemme relevante informationer via en database.

Technology and Data Variations List: Java, eclipse, GUI & JUnit testing, Maven, MYSQL Workbench, Sequel Pro og Windows 8 og 10, Macintosh IOS

ID 6: RollDice

Scope: Matador
Level: User goal
Primary actors: Player1-Player6.
Secondary actor: DTU (Ekkart og Bjarne)
Stakeholders and interests: Programmet skal kunne aflæse værdien af terningernes sum, samt aflæse hvornår en spiller slår to ens.
Preconditions: Spillere kan slå med terninger.
Success guarantee: Terningernes værdi vises, og værdierne er fuldkommen tilfældige for hvert slag.
Main Success Scenario (or Basic Flows):

1. En fra Player1-Player6 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 spiller slår to ens.

Extensions (or Alternative Flows):

3. Hertil gives spilleren en ekstra tur.
    a. Slår spilleren to ens igen, gives han/hun endnu en ekstra tur.
    b. Slår spilleren to ens på tredje slag, ryger den givne spiller i fængsel (includes 'Go to prison' use case).

ID 7: LandsOnProperty:

Scope: Matador
Level: User goal
Primary actors: Player1-Player6.
Secondary actor: DTU (Ekkart og Bjarne)

Stakeholders and interests: Spillerne skal gives mulighed for at købe felter, der ikke er chance-, fængsels eller parkeringsfelter. Ønsker en spiller ikke at købe, skal feltet ryge på auktion og giver alle mulighed for at byde og opkøbe det.
Success guarantee: Spillerne skal forholde sig til, om de vil byde eller ej.
Main Success Scenario (or Basic Flows):

1. En fra Player1-Player6 lander på et felt tilgængeligt for køb og køber det.
2. En fra Player1-Player6 lander på et opkøbt felt og betaler ejermanden af feltet.
3. En spiller ønsker at bygge på sin grund.

Extensions (or Alternative Flows):

1. Spilleren kan ikke betale ejermanden, når der landes på feltet.
    a. For at kunne betale ejermanden kan den givne spiller pantsætte egne grunde, huse og hoteller til banken, få penge herigennem og betale ejermanden efterfølgende. Spilleren kan også sælge sine værdier til feltets ejermand[16].
    b. Har spilleren fire huse på sin grund, kan spilleren nu vælge at bygge hotel på grunden i stedet.

ID 8: GoesToPrison
Scope: Matador
Level: User goal
Primary actors: Player1-Player6.
Secondary actor: DTU (Ekkart og Bjarne)
Stakeholders and interests: Programmet skal rykke en spiller som antal felter, som de summen er terningernes værdi, samt aflæse at en spiller slår to ens, hvortil han/hun får en ekstra tur.
Programmet skal kunne aflæse, når en spiller lander på et 'Go to prison felt' og hertil rykke spilleren til fængselsfeltet.
Preconditions: Spilleren kommer i fængsel, hvis han/hun lander på 'Go to prison'.
Success guarantee: Spilleren har flere muligheder til at komme ud af feltet.
Main Success Scenario (or Basic Flows):

1. En fra Player1-Player6 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-Player6 lander på et 'Go to prison'-felt.

Extensions (or Alternative Flows):

3. Spilleren rykkes til fængselsfeltet, fremfor at få muligheden for at rykke normalt.
    a. Spilleren gives først mulighed for at komme ud ved næste slag.
    b. Den givne spiller gives i denne situation ikke 4000kr, når han/hun passerer
    c. Spilleren kan vælge at betale 1000kr til banken for at komme ud.

[16] Se flowchart for ejendom

d. Spilleren får mulighed for at bruge et gemt chancekort, som kan få ham/hun ud af fængsel.
e. Spilleren får tre terningsslag for at komme ud. Slås der to ens, frigives spilleren og rykker samme antal, som terningerne viser.

10.2 Appendix 2 (Taksonomi)

Taksonomi

Field	Player	Mortgage
Chance card Account	Die	Game
Board	Real estate	Tokens (cars)
Prison	Bank	Color groups
Property	Hotel	House
Purchase Property	Pay rent	Purchase Real estate
Sell Real estate	Sell property	Salary(pass the start field)
Draw Chance card	Trade	

10.3 Appendix 3 (Syntaks for procedures)

Eksempel på proceduren "update_player".

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`update_player`(
        in playerID_input int(11),
        in gameID_input int(11),
        in position_input int(11),
        in prison_input int(11),
        in getOutPrison_input int(11),
        in balance_input int(11),
        in broke_input boolean,
        in current_input boolean
        )
BEGIN
UPDATE player
SET
    position = position_input,
    prison = prison_input,
    getOutPrison = getOutPrison_input,
    balance = balance_input,
    broke = broke_input,
    current = current_input
where (playerID, gameID) = (playerID_input, gameID);
END
```

10.4 Appendix 4 (Syntaks for view)

Eksempel på viewet "boughtproperties" med brug af Join ON.

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `boughtproperties` AS
    SELECT
        `property`.`fieldNumber` AS `fieldNumber`,
        `property`.`playerID` AS `playerID`,
        `property`.`houses` AS `houses`
    FROM
        (`property`
        JOIN `player` ON (((`property`.`gameID` =
`player`.`gameID`)
            AND (`property`.`playerID` =
`player`.`playerID`))))
```

10.4 Appendix 5 (Tests)

Test 2: realEstateTest

realEstate testen blev ikke succesfuldt oprettet, grundet sene ændringer i softwarens kode, den blev dog gennemført før dette, og stå i koden som udkommenteret.

Test Case ID	TC2
Referat	Automatiseret test af paySameTypeRealEstate. Begge spilleres pengebeholdninger testes.
Krav	Spillernes pengebeholdning bliver opdateret med det rette beløb
Betingelser før	Spillernes penge er 30.000
Betingelser efter	Spillernes penge er opdateret
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	100
Forventet resultat	30100 og 29900
Faktisk resultat	30100 og 29900
Status	Bestået
Testet af	Christian Høj
Dato	05/05-2018
Testmiljø	Eclipse Jee Oxygen på Windows 8

Test 4: moveToFieldTest

Test Case ID	TC4
Referat	Automatiseret test af moveToField
Krav	Spillerens position bliver opdateret til den nye position
Betingelser før	Spillerens position er 4
Betingelser efter	Spillerens position er opdateret
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	ny position=10
Forventet resultat	10
Faktisk resultat	10
Status	Bestået
Testet af	Christian Høj
Dato	05/05-2018
Testmiljø	Eclipse Jee Oxygen på Windows 8

Test 6: payMoneyTest

Test Case ID	TC5
Referat	Automatiseret test af payMoney
Krav	Spillerens pengebeholdning bliver opdateret med det rette beløb
Betingelser før	Spillernes penge er 30.000
Betingelser efter	Spillernes penge er opdateret
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	1000
Forventet resultat	29000 og 31000
Faktisk resultat	29000 og 31000
Status	Bestået
Testet af	Christian Høj
Dato	05/05-2018
Testmiljø	Eclipse Jee Oxygen på Windows 8