

The Deep Casino - Dokumentation

Projektgruppe 2



Deltagere

Danny Tran - 201610981: _____ Mehmetali Duman - 201611668: _____

Bekir Karaca - 201610458: _____ Stanie Cheung - 201607649: _____

Nina Nguyen - 201506554: _____ Muhanad Al-Karwani - 201611671: _____

David Lo - 201404972: _____ Frej Thorsen - 201611663: _____

Versionshistorik:

Version	Dato	Ændring	Initialer
0.0	12/09/2018	Oprettelse af rapport	SC & DT
0.1	26/09/2018	Ændret FURPS til at kunne testes	BK & DT
0.2	24/10/2018	Rettet kravspecifikation	Alle
0.3	13/11/2018	Analyse af Database	DT & MD

Indholdsfortegnelse

1	Indledning	5
2	Kravspecifikation	5
2.1	Funktionelle krav	5
2.1.1	MoSCoW	5
2.2	Ikke-funktionelle krav	6
2.2.1	FURPS	6
2.3	User Stories	6
2.4	Accepttest beskrivelse med brug af Gherkin:	7
3	Arbejdsfordeling	13
4	Metode og proces	14
5	Analyse	15
5.1	Database	15
5.2	Blackjack	15
5.2.1	Model-View-ViewModel	16
5.2.2	Static analysis og software matrice	18
5.3	Roulette	20
5.4	Opstartsapplikation	20
6	Softwarearkitektur	22
6.1	Domænemodel	22
6.2	Applikationsmodel	23
6.2.1	User story 3 - Blackjack	23
6.2.1.1	Klassediagram	23
6.2.1.2	Sekvensdiagram	24
6.2.2	User story 4 - Roulette	24
6.2.2.1	Klassediagram	24
6.2.2.2	Sekvensdiagram	25
6.2.3	User story 5 - Opstartsapplikation	26
6.2.3.1	Klassediagram	26
6.2.3.2	Sekvensdiagram	26
7	Softwaredesign	28
7.1	BlackJack	28
7.1.1	Klassediagram	28

7.1.2	Sekvensdiagram	29
7.2	Roulette	31
7.2.1	Klassediagram	31
7.2.2	Sekvensdiagram	32
7.3	Opstartsapplikation	33
7.3.1	Klassediagram	33
7.3.2	Sekvensdiagram	33
8	Software implementering	34
8.1	Database	34
8.1.1	CRUD operationer	36
8.2	BlackJack	40
8.2.1	SOLID	41
8.2.2	MVVM - BlackJack	41
8.2.3	Statisk analyse og Software matrice	44
8.3	Roulette	45
8.3.1	Private properties	45
8.3.2	spinButton_Click()	46
8.3.3	UpdateWallet()	47
8.3.4	GetBet()	47
8.3.5	Bet()	47
8.3.6	WinOrLose()	48
8.4	Opstartsapplikation	50
9	Test	52
9.1	Database - Test	52
9.1.1	CRUD operationer - Test	52
9.1.1.1	Create - Test	52
9.1.1.2	Read - Test	52
9.1.1.3	Update - Test	53
9.1.1.4	Delete - Test	53
9.1.2	Forbindelsesledet mellem databasen og DeepCasino - Test	53
9.2	BlackJack - Test	54
9.3	Roulette - Test	54
9.4	Opstartsapplikation - Test	55
10	Resultater	56
10.1	Database	56
10.1.1	Create	56

10.1.2 Read	57
10.1.3 Update	58
10.1.4 Delete	59
10.1.5 Forbindelsesledet mellem databasen og DeepCasino - Test	59
10.2 BlackJack	60
10.3 Roulette	63
10.4 Opstartsapplikation	66
11 Diskussion af resultater	68
11.1 Database	68
11.2 BlackJack	68
11.3 Roulette	69
11.4 Opstartsapplikation	69
12 Ordliste	70
13 Referenceliste	71
13.1 Database referenceliste:	71
13.2 BlackJack referenceliste:	71
13.3 Roulette referenceliste:	71
13.4 Opstartsapplikation referenceliste:	71

1 Indledning

Denne dokumentationsrapport vil beskrive selve arbejdsprocessen for gruppe 2, 4. semesters projektet. Rapporten vil komme ind på The Deep Casino's funktionaliteter, design, metoder, krav, resultater og diskussioner herom. Det vil give en dybere forståelse for selve projektet og teoretiske begreber, som er givet udtryk for i selve projektrapporten.

2 Kravspecifikation

2.1 Funktionelle krav

2.1.1 MoSCoW

Must have:

- Brugeren **skal** kunne indtjene deepcoins ved at vinde spil.
- Brugeren **skal** kunne få opbevaret deepcoins på sin konto i databasen.
- Brugeren **skal** kunne spille blackjack.
- Brugeren **skal** kunne spille roulette.

Should have:

- Brugeren **bør** kunne vælge mellem at spille BlackJack eller Roulette.
- Brugeren **bør** kunne oprette en brugerkonto selv.
- Brugeren **bør** kunne opleve visuel animation i spillene.

Could have:

- Brugeren **kan** have adgang til en virtuel butik (The DeepShop).
- Brugeren **kan** kunne bruge sine deepcoins på præmier i The DeepShop.
- Brugeren **kan** kunne læse reglerne for spillene i applikationen.
- Brugeren **kan** få applikationen fremvist på en HTML-hjemmeside.
- Brugeren **kan** få gem sin highscore for et vist spil.
- Brugeren **kan** kan opleve at spille har lydeffekter.

Won't have:

- Brugeren **kan ikke** interagere med andre brugere.
- Brugeren **kan ikke** benytte The Deep Casino som mobilapplikation.

2.2 Ikke-funktionelle krav

2.2.1 FURPS

Functionality:

- Windows-applikationen skal have en grafisk brugergrænseflade.
- GUI skal have et maksimum antal elementer/knapper på 6.
- Windows-applikationens elementer/knapper skal vise, hvilke formål elementerne/knapper har med tydelig tekst.
- Windows-applikationens sprog skal være engelsk.

Usability:

- Brugeren kan indstille størrelsen af det grafiske vindue.
- Brugeren kan indstille lydstyrken på lydeffekterne.
- Brugeren skal kunne vælge hvilket spil, brugeren vil spille.

Reliability:

- Brugeren kan inden for 14 dage fortryde sit køb på deepshoppen, så længe præmien ikke er udløst.

Performance:

- Det tager under 5 sekunder at starte applikationen op.
- Det skal tage ca. 2 sekunder for programmet at reagerer på events.

Supportability:

- The Deep Casino bør have vejledning til selve applikationen.
- I applikationen skal der være en knap hvori der indgår både Black jack- og Roulette-regler.

2.3 User Stories

Nedenfor ses user stories til spillet The Deep Casino, hvor hver user story har en prioritet, der går fra 1 til 10, hvor 1 er mindst prioriteret og 10 er højst prioriteret. Der er også en estimeret tid tilføjet, der oprettes i timer.

User Story	Priority	Estimate
1. Brugeren skal kunne indtjene deepcoins ved at vinde spil.	8	2t
2. Brugeren skal kunne få opbevaret deepcoins på sin konto i databasen.	9	18t
3. Brugeren skal kunne spille blackjack	10	18t
4. Brugeren skal kunne spille roulette	10	18t
5. Brugeren bør kunne vælge mellem at spille BlacJack eller Roulette.	7	7t
6. Brugeren bør kunne oprette en brugerkonto selv.	7	7t
7. Brugeren bør kunne opleve visuel animation i spillene.	5	8t
7. Brugeren kan have en adgang til en virtuel butik (The DeepShop).	5	8t
8. Brugeren kan kunne bruge sine deepcoins på præmier i The DeepShop.	4	8t
9. Brugeren kan kunne læse reglerne for spillene i applikationen.	4	3t
10. Brugeren kan få gemt sin highscore for et vist spil.	4	3t
12. Brugeren kan opleve at spillene har lydeffekter.	3	2t
13. Brugeren kan få applikationen fremvist på en HTML-hjemmeside,	4	7t
14. Brugeren kan ikke interagere med andre brugere.		
15. Brugeren kan ikke benytte The Deep Casino som mobilapplikation.		

2.4 Accepttest beskrivelse med brug af Gherkin:

EGENSKAB: 1. VIND DEEPCOINS

Som bruger af The Deep Casino,
 kan jeg vinde i de forskellige spil,
 så jeg kan indtjene virtuelle penge.

BAGGRUND:

Givet at brugeren er logget ind
 Og brugeren har deepcoins tilgængelig på account
 Og brugeren har følgende nuværende oplysninger:

Brugernavn	rolle
Brugernavn	Bruger

SCENARIO:

Når brugeren ønsker at tjene deepcoins
 Så vælger brugeren et spil
 Så satser han/hun en del af sine deepcoins
 Så trykker brugeren på start-knappen
 Så viser applikation at brugeren har vundet
 Og applikation adderer deepcoins på brugerens konto.

EGENSKAB: 2. GEM DEEPCOINS

Som bruger af The Deep Casino,
kan min konto og deepcoins blive gemt,
så jeg ikke mister dem ved at lukke applikationen.

BAGGRUND:

Givet at applikationen har en database tilknyttet,
Og at applikationen er funktionel med Account
Og at brugeren har en konto.

SCENARIO:

Når brugeren tjener, mister eller ingen af delene,
Så gemmes deepcoins i brugerkontoen, Så gemmes brugerkontoen i databasen.

EGENSKAB: 3. SPIL BLACKJACK

Som bruger af The Deep Casino,
kan jeg spille BlackJack,
så jeg kan blive underholdt og vinde deepcoins.

BAGGRUND:

Givet at brugeren er logget ind,
Og brugeren har deepcoins tilgængelig på account,
Og brugeren har følgende nuværende oplysninger:

Brugernavn	Rolle	Deepcoins
Brugernavn	Bruger	Nok

SCENARIO:

Når brugeren ønsker at spille BlackJack
Så vælger brugeren "BlackJack"
Så viser applikationen at BlackJack-spillet starter
Så satser brugeren en mængde deepcoins
Så starter spillet mod dealer
Så viser applikationen om brugeren enten har vundet eller tabt
Så adderer eller subtraherer applikationen deepcoins til/fra brugerens deepcoins.

EGENSKAB: 4. SPIL ROULETTE

Som bruger af The Deep Casino,
kan jeg spille Roulette,

så jeg kan blive underholdt og vinde deepcoins.

BAGGRUND:

Givet at brugeren er logget ind,

Og brugeren har deepcoins tilgængelig på account,

Og brugeren har følgende nuværende oplysninger:

Brugernavn	Rolle	Deepcoins
Brugernavn	Bruger	Nok

SCENARIO:

Når brugeren ønsker at spille roulette

Så vælger brugeren "Roulette"

Så viser applikationen at roulette-spillet starter

Så satser brugeren en mængde deepcoins på et udfald

Så viser applikationen et udfald

Så viser applikationen om brugeren enten har vundet eller tabt

Så adderer eller subtraherer applikationen deepcoins til/fra brugerens deepcoins.

EGENSKAB: 5. VÆLG SPIL

Som bruger af The Deep Casino,

kan jeg vælge at spille BlackJack eller Roulette,

så jeg kan spille det spil, som jeg har lyst til.

BAGGRUND:

Givet at brugeren er logget ind,

Og brugeren har deepcoins tilgængelig på account,

Og brugeren har følgende nuværende oplysninger:

Brugernavn	Rolle	Deepcoins
Brugernavn	Bruger	Nok

SCENARIO:

Når brugeren ønsker at spille

Så giver applikationen mulighed for at vælge BlackJack eller Roulette.

EGENSKAB: 6. OPRET KONTO

Som bruger,

kan jeg oprette en konto på The Deep Casino,

så jeg kan blive en bruger af The Deep Casino.

BAGGRUND:

Givet at brugeren kan tilgå applikationen.

SCENARIO:

Når brugeren ønsker at oprette en konto

Så åbner brugeren applikationen

Så indtaster han ønsket brugernavn

Så får han/hun adgang til applikationens funktionaliteter.

EGENSKAB: 7. VIS VISUEL ANIMATION

Som bruger af The Deep Casino,

Kan jeg opleve visuel animation i The Deep Casino,

Så jeg får en bedre oplevelse ved at bruge applikationen.

BAGGRUND:

Givet at brugeren er logget ind,

Og at brugeren har valgt et spil,

Og at brugeren har valgt et gyldigt beløb at satse.

SCENARIO:

Når brugeren i enten BlackJack eller Roulette trykker 'start' eller 'deal',

Så viser applikationsvinduet eksempelvis brugerens nuværende kort.

EGENSKAB: 8. BRUG THE DEEPSHOP

Som bruger af The Deep Casino,

kan jeg få adgang til The DeepShop,

så jeg kan bytte mine deepcoins deri.

BAGGRUND:

Givet at applikationen er funktionel

SCENARIO:

Når en bruger er logget ind

Så skal han/hun kunne vælge 'The DeepShop'.

Så viser applikationen at brugen befinder sig i The DeepShop.

EGENSKAB: 9. BRUG DEEPCOINS

Som bruger af The Deep Casino,

kan jeg bruge deepcoins i The DeepShop,
så jeg kan få præmier.

BAGGRUND:

Givet at brugeren er logget ind
Og at brugeren er inde i The DeepShop,
Og at brugeren har deepcoins tilgængelig på sin account,
Og at mængden af deepcoins er nok til den specifikke præmie,
Og at brugeren har følgende nuværende oplysninger:

Brugernavn	Rolle	Deepcoins
Brugernavn	Bruger	Nok

SCENARIO:

Når brugeren ønsker at bytte sine deepcoins for en præmie
Så vælger brugeren 'The DeepShop',
Så viser applikation de mulige præmier,
Så trykker brugeren 'køb' ud fra den ønskede præmie,
Så subtraherer applikationen deepcoins fra brugerens konto,
Så får brugeren en præmiekode fremvist.

EGENSKAB: 10. LÆS REGLER

Som bruger af The Deep Casino,
kan jeg i et spil læse om spillets regler,
så jeg kan lære, hvad spillet går ud på.

BAGGRUND:

Givet at brugeren er logget ind,
Og at brugeren enten har trykket 'BlackJack' eller 'Roulette'.

SCENARIO:

Når brugeren ønsker at læse om spillets regler,
Så vælger brugeren 'Rules',
Så åbner applikationen et separat vindue, hvori reglerne står,

EGENSKAB: 11. GEM HIGHSCORE

Som bruger af The Deep Casino,
Kan jeg gøre min highscore offentlig,
Så den kan ses af andre spillere.

BAGGRUND:

Givet at brugeren er logget ind,
Og har interageret med et spil i The Deep Casino,
Og brugeren har følgende nuværende oplysninger:

Brugernavn	Rolle
Brugernavn	Bruger

SCENARIO:

Når brugeren er færdig med en runde i et spil,
Så vælges 'Highscores',
Så angiver brugeren sin highscore,
Så trykker brugeren OK,
Så viser applikationen brugerens highscore,
Og applikationen tilføjer brugerens highscore til 'Highscores'.

EGENSKAB: 12. AFSPIL LYD

Som bruger af The Deep Casino,
Kan jeg opleve lydeffekter i applikationen,
Så jeg får en bedre oplevelse ved at bruge applikationen.

BAGGRUND:

Givet at brugeren er logget ind,

SCENARIO:

Når brugeren trykker på en tilfældig knap,
Så afspilles en lyd.

EGENSKAB: 13. ÅBEN HTML-HJEMMESIDE

Som bruger af The Deep Casino,
Kan jeg åbne applikationen på en HTML-HJEMMESIDE,
Så jeg kan tilgå applikationen gennem internettet.

BAGGRUND:

Givet at brugeren har en internetbrowser åben.

SCENARIO:

Når brugeren skriver den rette internetadresse ned,
Så åbnes The Deep Casino på en HTML-hjemmeside.

EGENSKAB: OPRETTE EN BRUGER-KONTO

Som bruger af The Deep Casino,
Vil jeg gerne oprette en konto,
Så jeg kan beholde mine virtuelle penge

BAGGRUND:

Givet at brugeren har interageret med applikationen
Og brugeren gerne vil oprette en konto,
Og brugere får følgende oplysninger:

Brugernavn	rolle
Brugernavn	Bruger

SCENARIO:

Når brugeren interagerer med applikationen,
Så skriver brugeren sine konto-oplysninger,
Så oprettes brugeren på systemet

3 Arbejdsfordeling

Måden hvorpå gruppen har opdelt sig, kan ses på tabel 1, som illustrerer fordelingen af de forskellige komponenter/-funktionaliteter af The Deep Casino til gruppemedlemmerne.

Arbejdsfordeling								
Opgaver	BK	DT	DL	FT	MD	MA	NN	SC
Database	X	X			X			
BlackJack			X			X		
Roulette				X				X
Opstartsapplikation			X			X	X	

Tabel 1: Tabel over arbejdsfordelingen i gruppen.

4 Metode og proces

Som fremgangsmetode for arbejdet bruger gruppen SCRUM i det at opgaverne har været inddelt i sprints, hvilket kan ses i gruppens referater. Det er ikke SCRUM i den klassiske forstand at gruppen har arbejdet ud fra. Det er en tilpasset version. De elementer fra SCRUM der er brugt er sprint-planning, sprint-review og sprint-retrospective. Disse SCRUM elementer er diskuteret ved hvert møde siden gruppen blev opdelt. Der er blevet holdt styr på de forskellige sprints i mødereferaterne. Gruppen har haft en fast referent til at notere sig alle aktionspunkter fra møderne. Gruppen overvejede forskellige måder at holde sprint-forløbene overskuelige for alle. Gruppen fandt derefter frem til, at dette sagtens kunne gøres fyldestgørende gennem mødereferaterne, da alle medlemmer aktivt har været inde på gruppens repository og kigge på aktuelle referater.

Fremgangsmåden har været på den måde at gruppen har delt sig i mindre grupper for at udføre de forskellige sprints. Det er alt efter arbejdsfordelingen. Sprint-grupperne stod til ansvar for at de tildelte sprints blev ordnet. Under hele processen forekom frihed i henhold til planlægningen af sprints, fordi at gruppen samtidig arbejder sig frem på en iterativ måde. Det giver plads til at træde tilbage i arbejdet, når der sker en potentiel fejl. På trods af at sprint-grupperne har fordybet sig i hver deres dele, har alle gruppemedlemme haft overblik over hele projektet. Gruppen har været fælles om at udarbejde problemformuleringen, projektbeskrivelsen og kravspecifikationen for det overordnede produkt.

5 Analyse

I dette afsnit beskrives de overvejelser om mulige løsninger gruppen har haft til hhv. Database-, opstartsapplikationen-, BlackJack- og Roulettedelen. Derudover også de løsninger gruppen har valgt og begrundelsen dertil.

5.1 Database

Da det er et krav at have en database i projektet blev det fastlagt, at DeepCasino skal have en database med brugerinformationer, hvori brugeren indtaster et username, som derefter skal skrives ind i databasen vha. CRUD operationerne. Databasen bliver udarbejdet på udviklingsværktøjerne, Visual Studio og DDS-Lite, hvori der bliver inddraget viden fra E18I4DAB omkring relationelle databaser. Databasen bliver også udarbejdet på en server database, som derefter skal forbindes til de andre projekter.

Til DeepCasino spil, har gruppen brug for at kunne gemme data af brugeren, såsom username, wallet og highscore. Det er nødvendigt at have en database til The Deep Casino spil, da den skal gemme brugerens username og highscore, dermed sammenligne med de andre brugere. Det blev i starten af projektet diskuteret, at databasen skulle indeholde både brugernavn og et kodeord. Men efter flere overvejelser og samtaler med vejlederen, blev det fastlagt, at man kun skal have et brugernavn, eftersom det blev for kompliceret med et kodeord.

Databasen er en relationel database uden benyttelse af Swashbuckle samt Azure Cosmos DB, hvilket er nogle værktøjer til en database som en webapplikation. Grunden til gruppen ikke bruger Swashbuckle eller Azure er fordi, at første emne i E18I4DAB om databaser var relationelle databaser. Så gruppen gik straks igang med at lave en relationel database. Eftersom gruppen var færdig med den relationelle database, følte gruppen, at databasen ikke skulle udvides. Men efter gruppen fandt ud af, at der var flere måder at lave en smartere database på i E18I4DAB, havde gruppen ikke tid nok til at kunne udvide databasen. Databasen kører på SQL-serveren "st-i4dab.uni.au.dk", som er oprettet via. underviseren i I4DAB - kurset. Når man kører programmet og dermed opretter en bruger i applikationen, så vil man være i stand til at kunne se den data fra en anden enhed. Tabellen i databasen er man i stand til at kunne se brugerens username, highscore og wallet (DeepCoins).

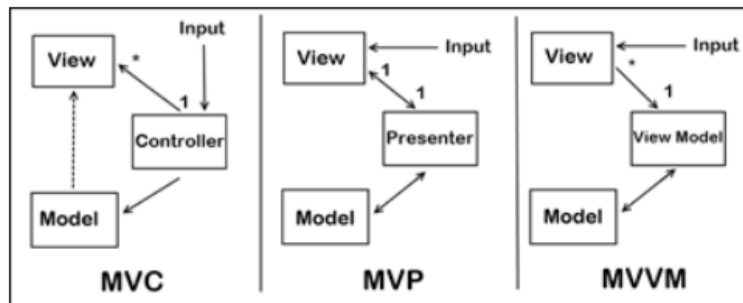
5.2 Blackjack

BlackJack er blevet udført med en masse overvejelser i forhold til hvilket af SOLID designprincipperne og design patterns, der vil passe bedst til gruppens applikation. Yderligere har der været overvejelser om hvordan gruppen har skulle anvende software test til, at verificerer og komme softwarekvalitetsproblemer i forkøbet i den implementerede kode. Sluteligt har gruppen ligeledes overvejet, hvordan userinterfacet for BlackJack applikation har skulle se ud samt interagerer med det bagvedliggende kode.

softwarekvalitetsproblemer i forkøbet, har gruppen bestemt sig løbende at bruge static analysis. Static analysis giver et overblik over disse problemer. Yderligere beskrivelse og forklaring til static analysis findes i afsnittet "Static analysis og software matrice".

5.2.1 Model-View-ViewModel

Gruppen har overvejet tre forskellige design pattern, der er forholdsvis tæt forbundet da de er en udbyggelse af hinanden. Valgt drejer sig henholdsvis om MVC(Model - View -Controller), MVP(Model - View - Presenter) og MVVM(Model - View - ViewModel), der alle tre går under kategorien som GUI design patterns. Valget ligger imellem disse design patterns, da de beskæftiger sig med effektivisering af applikationer, der gør brug af grafisk brugergrænseflader, hvilket netop er tilfældet med gruppens applikation.



Figur 1: Maintainability metric

Figur 1 er et klassediagram, som illustrerer forskel og ligheder imellem de tre design pattern. En kort gennemgang af MVC samt MVP vil blive beskrevet for, at give den optimale forståelse for gruppens valg af design pattern. Efterfølgende vil der komme en mere detaljeret beskrivelse af MVVM, da dette er gruppens valg af design pattern.

MVC design pattern er ligesom de to andre design pattern bestående af tre domænemodeller "Model", "Controller", og View. Model har i alle tre design patterns til formål at holde på applikationens bagvedliggende kode, hvilket også bliver kaldt "Business logic". Med andre ord så er models ansvarlig for data af applikations domænet. Controller er det centrale i MVC da denne klasse har til formål, at evaluere brugerinput og finde ud af hvad det betyder for modellen. Ydermere har Controller til formål at, hvilket View som skal vises. View udgør sammen med Controller præsenterationsdelen af MVC, og har netop til formål præsenterer modellen stadie og data til brugeren, som interagerer med View.

MVP design pattern er ikke så forskellige fra MVC design pattern, hvilket ligeledes kan ses i klassediagrammet. Model og view har i MVP samme funktionalitet, som er i MVC og gruppen vil derfor ikke nævne disse igen. Forskellen ligger dog i Presenter klassen, der er blevet erstattet med Controller klassen. I dette pattern har view og model ingen form for kendskab til hinanden og presenter virker derfor som et mellemlid mellem view og model.

MVVM design pattern er en specialisering af MVP, der er udviklet af Microsoft og anvendt i WPF applikationer. Dette design pattern er blevet udviklet til at anvende specifikke funktioner i WPF, hvor primært der bliver brugt data binding, til bedre udnytte adskillelsen af View udviklingen ved at fjerne alt business logic fra views. I Stedet for at kræve designere at skulle skrive deres view logik, kan der nu anvendes XAML til at oprette bindings til viewmodel, som er skrevet og vedligeholdt af applikationsudviklere. Det ses udefra klassediagrammet at view er forbundet med viewmodel, hvilket netop er muligt med data binding, der sender kommandoer til viewmodel. Viewmodel

interagerer med model gennem properties, metodekald samt ved modtagelse af events fra model. Model kender ikke til viewmodel. Viewmodel kender ikke til view, hvilket er den primære årsag til den løse kobling i dette design pattern.

Modellen er ligesom i MVC/MVP en objektmodel, som er en repræsentation af den reelle tilstand. View omfatter alle elementer, som bliver vist af den grafiske brugergrænseflade, dette omfatter vinduer, knapper, grafik og andre kontroller.

Et view kan repræsentere et helt vindue eller dele af et vindue. Viewmodel er en model af viewet i den forstand at det er en abstraktion af viewet, da denne er bindeleddet mellem viewet og modellen. Viewmodel kan ses som et specialiseret aspekt af presenter, som agerer data-binder/converter, der ændrer modellens informationer til view informationer og dirigerer commands fra viewet ned til modellen. Kommunikation mellem et view og dets viewmodel foretages hovedsageligt gennem data binding, hvor et view DataContext property indeholder en reference til viewmodel.

Processen i at binde præsentationslaget direkte til properties i en viewmodel er relativt ligefrem, men det at binde brugerinput til metoder i viewmodel virker ikke. I WPF kan den indbyggede Command Routing ikke nå viewmodel. Der findes en række løsninger på denne problemstilling. De to vi er blevet præsenteret for er "RelayCommand" og "DelegateCommand". Begge løsninger udnytter det faktum at command properties i WPF tillader brugen af command Types i stedet for RoutedCommand. Det påkræves at implementere ICommand interfacet for at implementere Commands. Den foreslåede fremgangsmåde lyder: tilføj Command properties til viewmodel, map/bind derefter commands metoder i viewmodel. Bind derefter command properties i viewmodel til command properties i WPF Controls.

Ulempen ved MVVM design pattern ligger i, at der er behov for en fortolkning mellem Viewmodel og view, og dette kan have repræsentationsomkostninger. Komplexiteten af denne fortolkning kan også variere: det kan være lige så enkelt som at kopiere data eller så kompleks som at manipulere det til en formular, vi gerne vil se i den grafiske brugergrænseflade. Dette er ikke tilfældet for MVC, hvor view kender til model.

De tre design patterns er alle effektive når der bliver arbejdet med grafiks brugergrænseflader med det formål af dele applikationen så komponenterne holder på deres eget ansvar.

Gruppen har dog valgt at anvende MVVM design pattern da dette pattern har fordele, der ikke kan gengives i MVC og MVP. En af disse fordele ligger i, at MVVM lægger op til at få 100% coverage i test af applikationen via unittest. Dette skyldes at Viewmodel samt model ikke har nogen form for afhængighed til View, og derfor vil det være nemt at substituere.

Yderligere vil brugen af MVVM give gruppen den fordel, at det kan implementeres i samtlige øvrige applikationer idet det ikke er begrænset til WPF-applikationer for windows desktop. Dog er dette ikke gældende for View, som skal bygges til den specifikke platform man ønsker at anvende. Som nævnt tidligere er Viewmodel og models domænemodellen ikke afhængige af view og disse kan derfor blive anvendt i samtlige andre applikationer. I modsætning til MVP's presenter så behøver viewmodel i MVVM ikke at have en reference til view. Dette resulterer i at design udvikleren i applikationen ikke behøver at have nogen form for kendskab til udvikleren bag business logic.

5.2.2 Static analysis og software matrice

Gruppen har i udarbejdelse af kode overvejet samtlige metoder, der kan sikre kvalitetssikret software. Der har været undersøgelser af metoder der interagerer med det anvendte programmeringsværktøj (Visual Studio), hvori source-koden bliver udarbejdet. Der vil blive anvendt statisk analyse under udarbejdelsen af BlackJack. Statisk analyse er en metode indenfor softwareudvikling, som bliver anvendt til at eksaminere kode uden at køre programmet. Med andre ord bygger dette koncept på, at give feedback til programmøren om hvordan kodestrukturen forholder sig uden at eksekvere programmet. Statisk analyse har altså fokus på analyse af kodens kompleksitet og ikke dens funktionalitet.

Gruppen gøre brug af primært to værktøjer til statisk analyse, som er Visual Studio og ReSharper. Disse værktøjer giver begge muligheden for køre kvantitativ analyse, som er et andet ord for software matrice. Software matricen giver os fem faktorer til at definere kodestrukturen og disse faktorer kan se på figur 39.

Maintainability index er et indeks, som beskriver hvor holdbar og effektiv ens kode er, samt hvor let den er at vedligeholde. Indekset går fra 0-100, hvor 100 er det bedste resultat. En generel skala der går igen blandt flere SW-udviklingshjemmesider ses på figur 2. Det som skalaen indikerer, er hvilket niveau der er tilstrækkeligt før kode kaldes holdbart.

Low Maintainability	Moderately Maintainability	Good Maintainability
0-9	10-19	20-100

Figur 2: Maintainability metric

Det ses ud fra skalaen (figur 2) at et maintainability index mellem 0-9 er en indikation på dårlig holdbarhed i koden. Indikationer vil også fremstå i Visual Studio, hvor low maintainability vil blive angivet med en rød cirkel. En gul trekant vil indikerer at koden er moderat holdbar og har maintainability index imellem 10 og 19. Alt over 19 vil blive indikeret med en grøn firkantet blok, og er godt holdbar kode.

Maintainability index giver et udmærket billede af, hvilket kode der skal gøres opmærksom på. Det er dog vigtigt, at man er opmærksom på øvrige parameter idet indekset er en udregning, der gør brug af tre øvrige parameter, lines of code, cyclomatic complexity og Halstead Volume. To af disse parametre kan ligeledes ses på figur 39, og hvis der ikke bliver taget hånd om dem, får man ikke det fulde ud af statisk analyse. Gruppen vil i dette afsnit gå i dybden med "Lines of code"og "Cyclomatic Complexity"og en beskrivelse af "Halstead Volume"vil kun findes i dokumentationen under "Analyse".

Lines of code angiver antallet af intermediate language-linjer(sprog mellem C og maskinkode) i koden dvs. det er ikke linjer i source koden. Dette anvendes til at indikere om en metode har for meget ansvar og skal deles op i flere mindre metoder. Det kan være svært at vedligeholde en lang metode. Her har gruppen anvendt single responsibility principle til at dele ansvaret af funktioner ud i mindre funktioner.

Halstead Volume bliver brugt til at angive størrelsen på implementationen af en algoritme. Udregning af Halstead Volume er baseret på antal af operationer udført og operander, som bliver håndteret i algoritmen.

Cyclomatic Complexity anvendes til at angive kompleksiteten af et program. Den giver et tal af, hvor svært koden er at teste, vedligeholde, fejlfinde og hvor sandsynligt det er at koden vil lave fejl. Med Cyclomatic Complexity går man efter, at få et lavt tal, da dette tal, afgøre hvor mange test cases, der er nødvendige for den enkelte funktion for at få 100 % coverage. Formlen for udregning af Cyclomatic Complexity:

$$M = E - N + 2 * p$$

Hvor,

$$E = \text{NumberOfEdges}$$

$$N = \text{NumberOfNodes}$$

$$P = \text{Thenumberofconnectedcomponents}$$

Til udfyldning af formelen for Cyclomatic Complexity har gruppen gjort brug af et flow chart diagram. I implementerings afsnittet vil gruppen give et eksempel samt forklaring på hvordan Cyclomatic complexity er blevet anvendt i kode og hvordan den er udregnet[ref].

Software matricen består af to øvrige parameter, Depth of inheritance samt Class coupling. Depth of Inheritance angiver antallet af klassedefinitioner, der strækker sig til roden i klassehierarkiet. Desto dybere hierarkiet er, jo vanskeligere kan det være at forstå, hvor bestemte metoder og properties er defineret.

Class coupling angiver koblingen til unikke klasser gennem parametre, lokale variabler, returtyper, metodekald, basisklasser, interface implementeringer mm.. God software design dikterer, at typer og metoder skal have "high cohesion" og lav kobling. Høj kobling indikerer et design, der er vanskeligt at genbruge og vedligeholde på grund af dets mange indbyrdes afhængigheder på andre typer.

Der er ting, som statisk analyse ikke kan identificere. Statisk analyse kan f.eks. ikke registrere om eventuelle softwarekrav er opfyldte, eller hvordan en funktion vil eksekveres. I sådan et tilfælde skal der bruges dynamisk analyse. Derfor er statisk analyse og dynamisk analyse komplementære. Statisk analyse opdager fejl i kode tidligt. Dette sikrer at et produkt af højere kvalitet når testfasen. Dét fremskynder udviklingen ved at sikre, at testprocesserne er mere effektive.

5.3 Roulette

Udviklingen af roulette-applikationen til The Deep Casino har haft mange overvejelser over sig, for at finde netop den løsning som gruppen fandt mest hensigtsmæssig, både i forhold til sværhedsgrad og kravene om indragelse af fagene på 4.Semester.

Da forholdene omkring roulette skulle besluttes, satte gruppen sig ind i, hvordan et moderne roulette spil fungerede og hvilke funktionaliteter det havde. Med udgangspunkt i søgen på nettet og hjælp fra undervisningen gjorde gruppen sig nogle tanker.

Den første tanke der strejfede gruppen, var at koden skulle skrives i sproget C#, da dette var det grundlæggende sprog for dette semester. Derudover blev det også fastlagt at simplificere spille reglerne for Roulette, hvor der kun vil være fokus på de hovedsaglige spil såsom ulige-tal, lige-tal, sort, rød og grøn. Yderligere blev der taget op, om roulette-hjulet skulle visualiseres. Det blev hurtigt afgjort, at opgaven for det visuelle ville blive for tidskrævende.

Det blev yderligere diskuteret, at lave applikationen i forbindelse med ét af de mange design patterns fra faget I4SWD. For at fastholde kodestrukturen konsistent med Blackjack-applikationen, blev det fastlagt at anvende MVVM. Dette vil også gøre unit testen og integrationstesten på applikationen meget nemmere, grundet opdelingen mellem de forskellige lag.

Da det øvrige afsnit af Blackjack allerede har beskrevet det overordnede teori omkring MVVM, er det ikke blevet taget hånd om her.

5.4 Opstartsapplikation

I dette afsnit beskrives gruppens tanker bag designet og implementeringen af applikationen, hvori BlackJack og Roulette kommer til at være inkluderet.

Opstartsapplikationen har til formål at gøre det muligt for brugeren at logge ind med et brugernavn og tilgå spillene BlackJack og Roulette. Dette er vigtigt, da gruppen har en User Story (US 5), der baserer sig på at brugeren skal kunne vælge hvilket spil, han/hun ønsker at spille. Da denne del ikke er en af de øverst prioriterede faktorer ift. gruppens User Stories, har gruppen bestemt at denne del af projektet ikke er beskrevet detaljeret i rapporten. En beskrivelse af opstartsapplikationen af The Deep Casino findes i gruppens dokumentationen.

Gruppens forestilling omkring opstartsapplikationen har været at holde den simpel, men fuld funktionel, netop fordi at den ikke har stor betydning ift. gruppens funktionelle krav.

Denne applikation er præget af teorien som gruppen har tillært sig i kurset I4GUI (GUI programmering), specielt inden for emnerne:

- .Net og C#.
- WPF, XML og XAML.
- Layout, Commands, Keyboard og Mouse input.

- Controls, The Application, Menus og toolbars.

For at løse denne delopgave, har gruppen tænkt sig at bygge en WPF-applikation som lært i GUI. WPF-applikationen gøre det overskueligt for gruppen som udviklere, at kunne lave program, der kan skifte mellem sider/vinduer, således at man som bruger ikke har mere end ét vindue åbent.

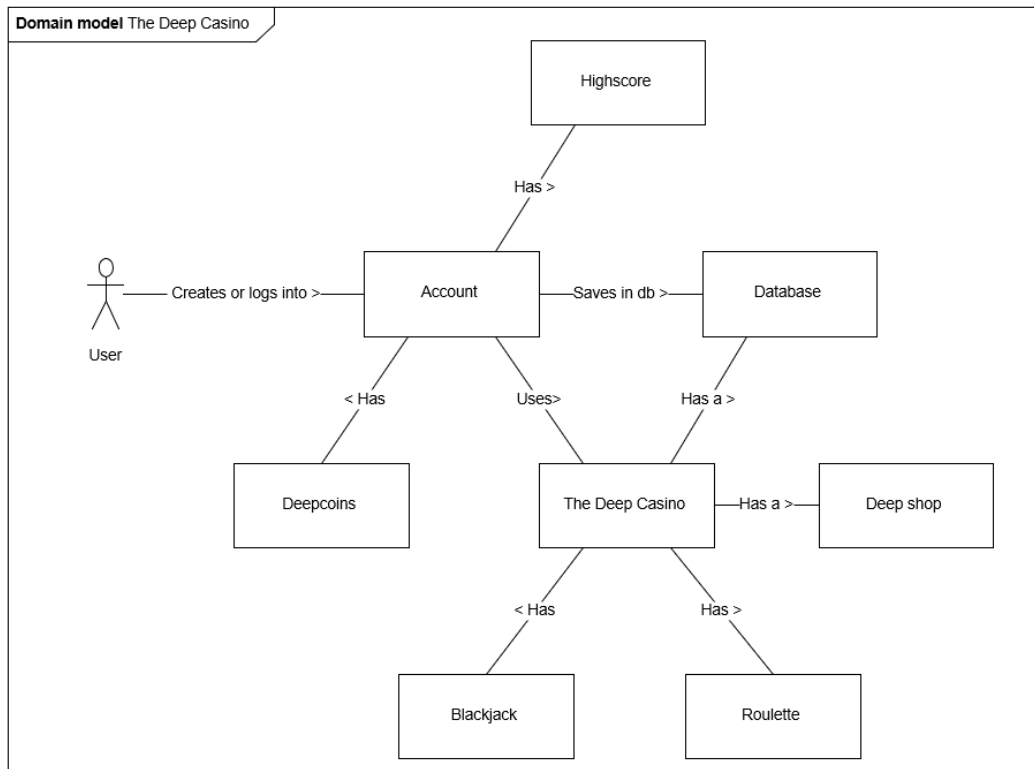
En essentiel model, som gruppen tænker at anvende i forbindelse med nævnte funktionalitet i denne del, er modellen ContentControl. Denne model høre under et bredt emne inden for GUI kaldet The Content Model. ContentControl er en model som bruges ift. funktionalitet i en WPF-applikation. Mere specifikt repræsenterer denne model styring som kan indeholde en enkel content-enhed. Brugen af ContentControl, vil gøre det lettere for gruppen at kunne skifte mellem vinduer.

6 Softwarearkitektur

Dette afsnit har til formål at beskrive The Deep Casino's softwarearkitektur. Her indgår en udarbejdet domænemodel, der dækker The Deep Casino på baggrund af User Stories. Endvidere fremvises et udarbejdet applikationsmodel med generelle klassediagrammer uden attributter, samt sekvensdiagrammer uden funktionskald. Til sidst fremvises et rekonstrueret applikationsmodel, hvor klassediagrammer indgår attributter, og sekvensdiagrammer indgår funktionskald. Dette vil blive fremvist under afsnittet Softwaredesign. Der er valgt at lave et applikationsmodel på de tre mest betydende User Stories, for at vise det principielle af The Deep Casino.

6.1 Domænemodel

På figur 3 nedenfor, ses domænemodellen. Det ses at brugeren enten opretter eller logger ind på brugerkontoen. Hvis det er tilfældet at brugeren opretter en ny konto, vil den blive oprettet og gemt i databasen. Endvidere får den nyoprettede konto tildelt en highscore og deepcoins. Kontoen får tilgængelighed til The Deep Casino, hvor brugeren har mulighed for at vælge mellem tre forskellige instanser, som henholdsvis er Deep shop, Blackjack og Roulette. Deep shop viderefører brugeren til en butiksliste, hvor brugeren har mulighed at anvende deepcoins på præmier. Blackjack åbner op for spillet BlackJack, og roulette åbner op for spillet Roulette.



Figur 3: Domænemodel af systemet

6.2 Applikationsmodel

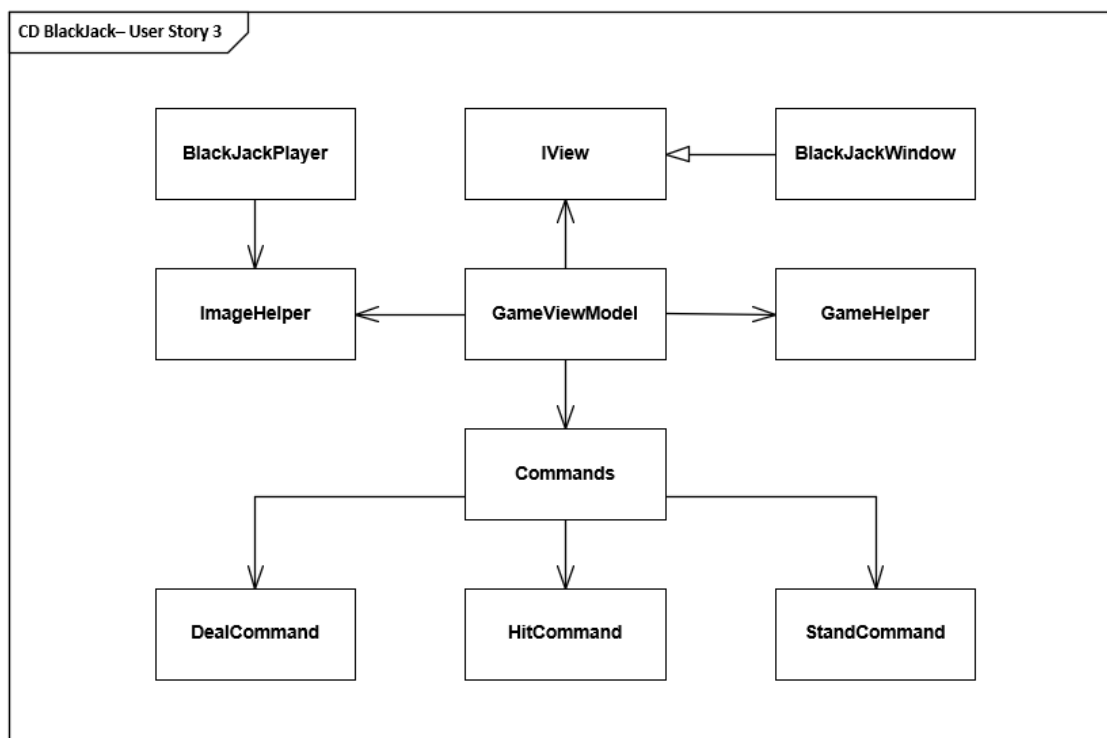
Med et indblik på domænemodellen, er det næste aktionspunkt at udarbejde applikationsmodellen. Tidligere blev der nævnt at lave et applikationsmodel på de tre essentielle User Stories for The Deep Casino. Indledningsvist er der lavet generelle klassediagrammer for User Story 3, User Story 4 og User Story 5. Endvidere er der udarbejdet sekvensdiagrammer baseret på klassediagrammerne, som vil beskrive funktionaliteten og samarbejdsen for de tilhørende klasser.

6.2.1 User story 3 - Blackjack

I dette underafsnit beskriver gruppen BlackJack-systemet med fokus på opdeling af systemet i klasser og hvorledes disse klasser kommunikere indbyrdes.

Figur 4 er en overordnet illustration i form af et klassediagram over gruppens sammensætning af nødvendige klasser. Gruppen har valgt at hele programmet køre afhængigt af GameViewModel, i den forstand at det er den klasse, der bruger alle de andre essentielle dele af systemet, hvilket også vises ved associationspilene mellem klasserne på diagrammet.

6.2.1.1 Klassediagram

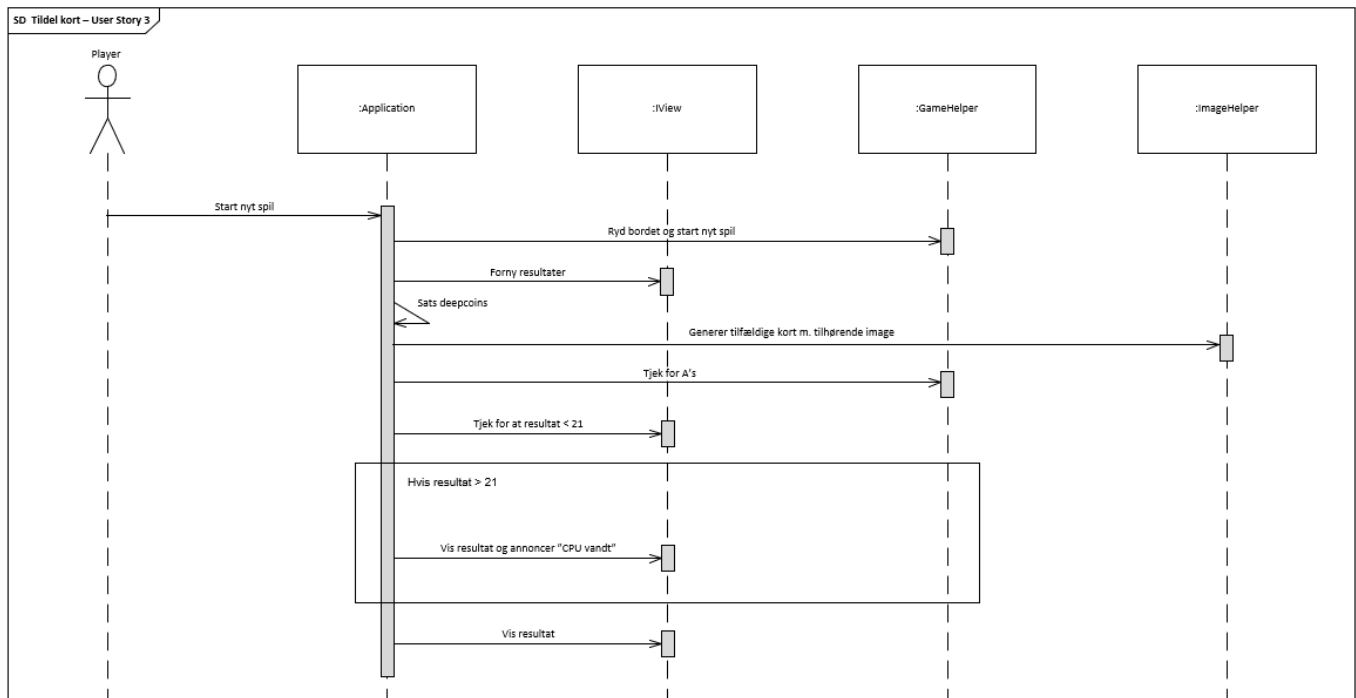


Figur 4: Klassediagram for hele BlackJack-applikationen

På sekvensdiagrammet på figur 5, illustrerer gruppen, hvorledes kommunikationen mellem klasserne i store træk kommer til at foregå. Gruppen har vha. beskrivende tekst over pilene, beskrevet hvorledes klasserne spiller sammen og i hvilken rækkefølge.

Klassen Application er en sammensætning af GameViewModel, Commands og de klasser som Commands bruger (fra klassediagrammet på figur 4).

6.2.1.2 Sekvensdiagram



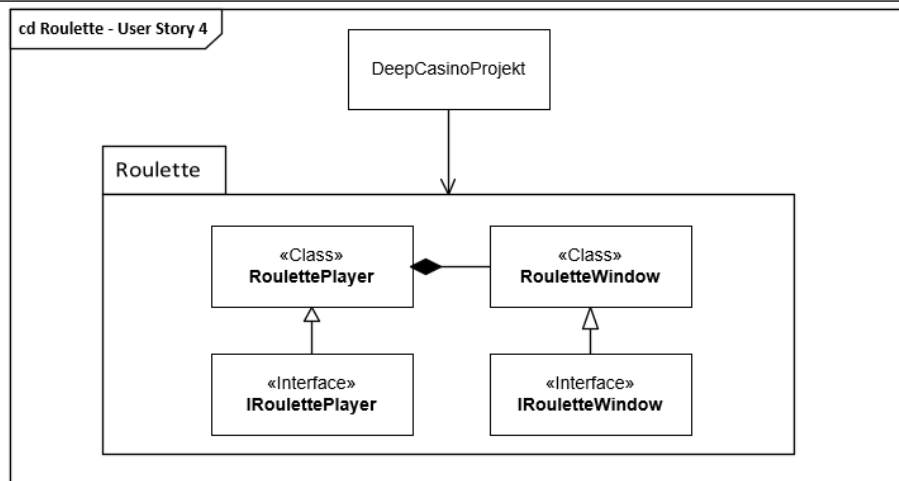
Figur 5: Sekvensdiagram for start af BlackJack

6.2.2 User story 4 - Roulette

I dette underafsnit behandles User Story 4, "Spil Roulette". Her fremgår først et klassediagram uden members, efterfulgt af et sekvensdiagram uden funktionskald, der viser en handling af brugeren der satser ti deepcoins på farven sort.

6.2.2.1 Klassediagram

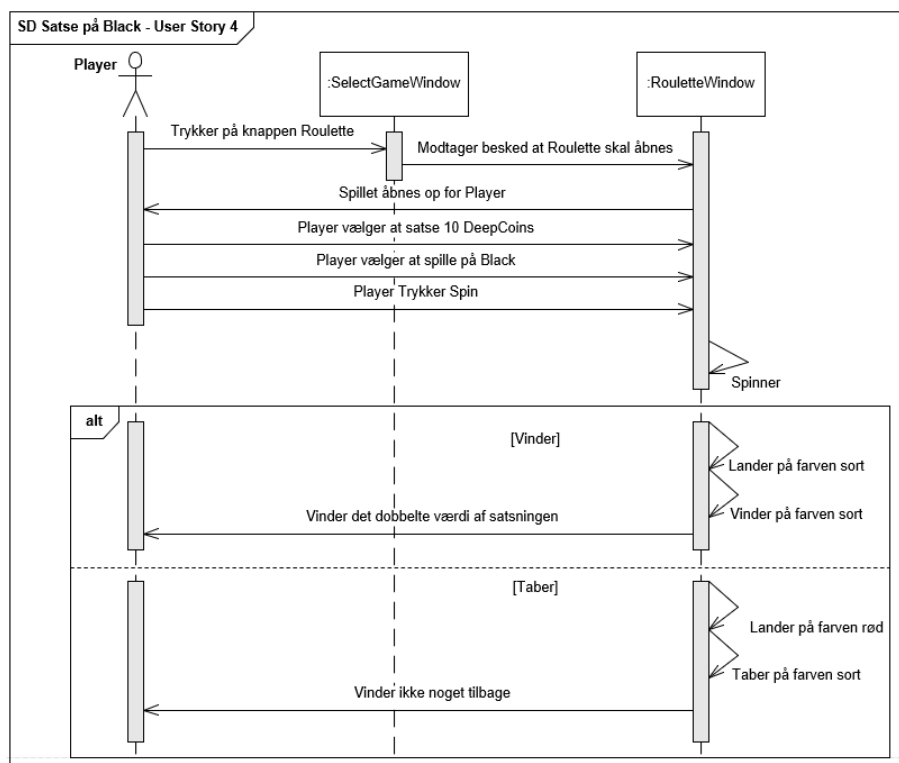
Det ses på figur 6 nedenfor, hvordan samarbejdet mellem klasserne er opstillet. Pilene viser hvordan blokkene er associeret med hinanden. I skitsen kaldet "Roulette" ses fire blokke. Disse skal ses som applikationen for spillet roulette. Det ses også at en blok kaldet DeepCasinoProjekt er associeret med skitsen Roulette, denne blok er Opstartsapplikationen, som ses på figur 8 i underafsnittet User Story 5 - Opstartsapplikation. Overordnet viser diagrammet kommunikationen af Roulette-applikationen.



Figur 6: Klassediagram for hele Roulette-applikationen

6.2.2.2 Sekvensdiagram

Det er valgt at vise en enkel sekvens af hele Roulette-applikationen, på baggrund af deres ensartede scenarier. Dette vil overordnet vise hvordan applikationen udfører et enkelt spil af roulette. Figur 7 nedenfor viser en sekvens af en spiller, der satser på farven sort i Roulette-applikationen.



Figur 7: Sekvensdiagram for satsning af farven sort på Roulette-applikationen

Til start trykker Player på knappen Roulette, som vil starte spillet roulette. Player vælger at satse ti deepcoins på farven sort. Dernæst trykker Player på knappen Spin, og spillet kører. Afhængig af hvad resultatet vil blive, kan

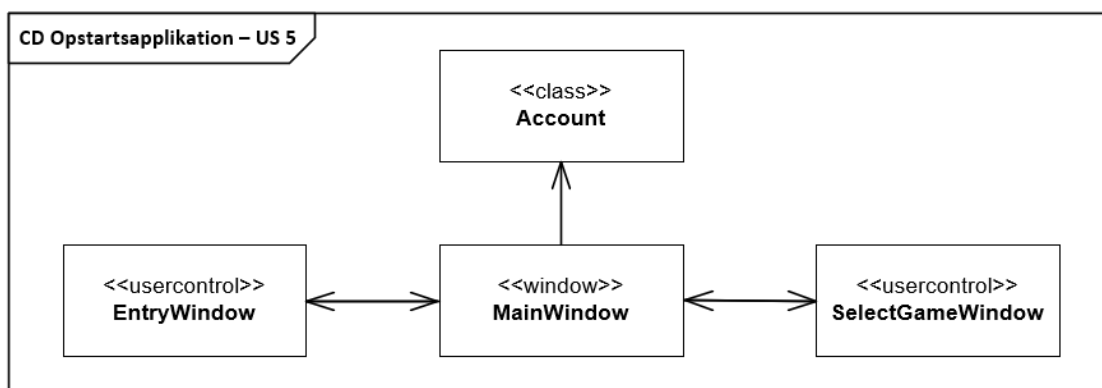
Player enten vinde eller tabe. Hvis det er tilfældet at der landes på farven sort, vinder Player og bliver belønnet med det dobbelte beløb Player har satset. Hvis det er tilfældet at der ikke landes på farven sort, taber Player og bliver ikke belønnet.

6.2.3 User story 5 - Opstartsapplikation

Dette afsnit indeholder diagrammer over gruppens opstartsapplikation. I forhold til gruppens domænemodel på figur 3, svarer denne del til "The Deep Casino-blokken.

6.2.3.1 Klassediagram

På figur 8 illustreres hvordan gruppen har bygget og sammensat de forskellige vinduer og klassen Account. MainWindow bliver kørt så snart at applikationen startes og giver ikke i sig selv noget visuelt billede, men den sørger for at EntryWindow aktiveres.

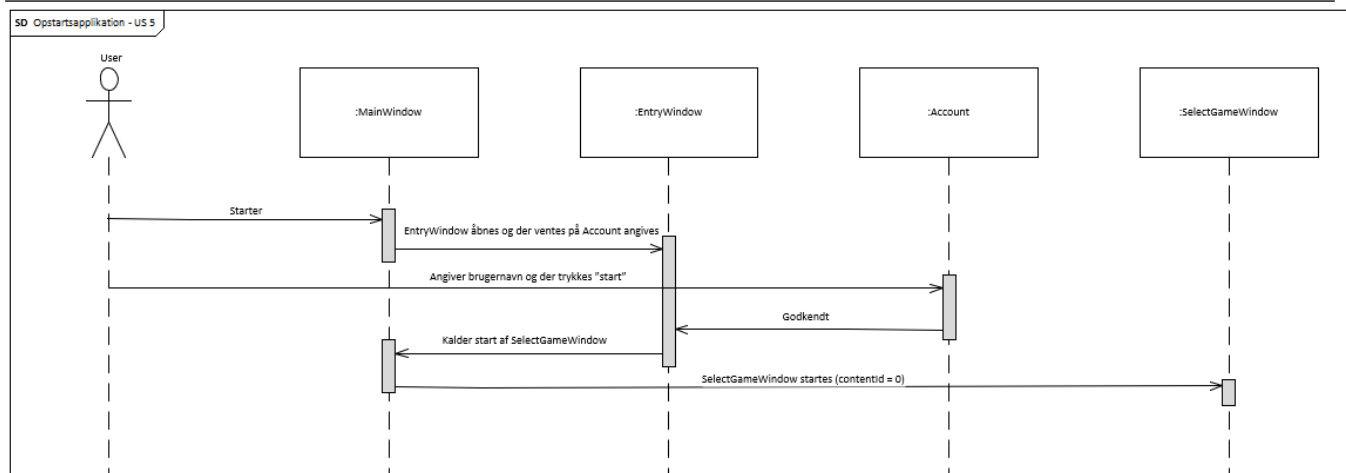


Figur 8: Klassediagram for opstartsapplikationen.

Account fra figur 8 er en klasse i applikationen, hvori brugerens informationer skal gemmes. EntryWindow er vinduet, som bliver åbnet så snart at applikationen køres. Selve vinduet som brugeren ser, skal indeholde en Start-knap og et tekstfelt, hvori brugeren kan indtaste sit brugernavn. SelectGameWindow aktiveres efter at brugeren er logget ind gennem EntryWindow, og deri kan brugeren vælge hvilke spil, der skal åbnes.

6.2.3.2 Sekvensdiagram

På figur 9 ses et sekvensdiagram til illustrering af hvorledes opstartsapplikationen køre fra at brugeren starter applikationen til at han/hun når til SelectGameWindow.



Figur 9: Sekvensdiagram over opstart af The Deep Casino-applikationen.

7 Softwaredesign

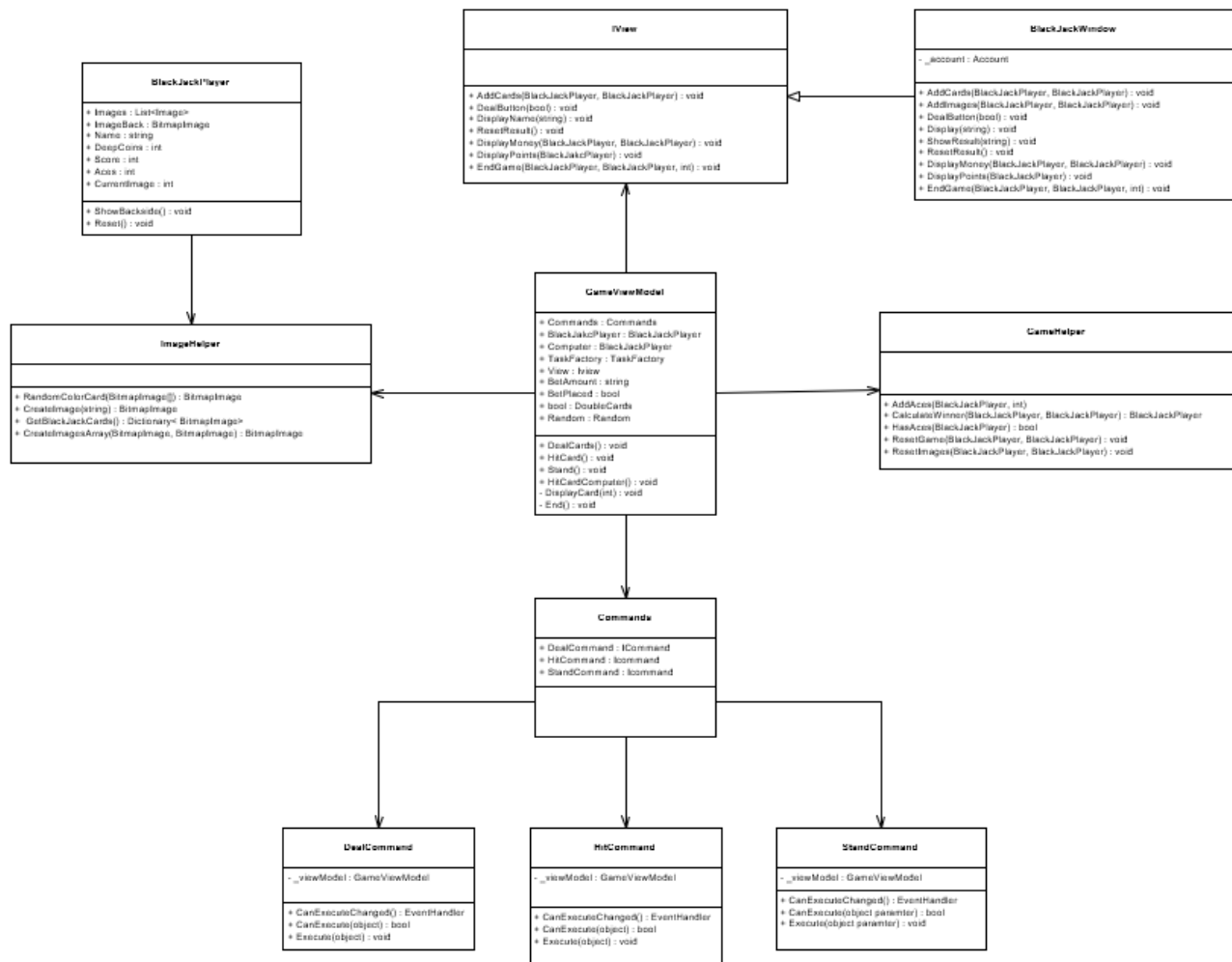
Der har i projektgruppen arbejdet med at få The Deep Casinos funktionaliteter på plads. Følgende punkter forneden beskriver henholdsvis softwaredesignet for Databasen, opstartsapplikationen og implementering af de to Casino-spil; BlackJack og Roulette.

7.1 BlackJack

I dette underafsnit beskrives den interne funktionalitet i de klasser som er beskrevet under "BlackJack" i afsnittet softwarearkitektur. Dette er grunden til at gruppen har inkluderet et mere detaljeret klasse- og sekvensdiagram med attributter og funktionskald i dette afsnit.

7.1.1 Klassediagram

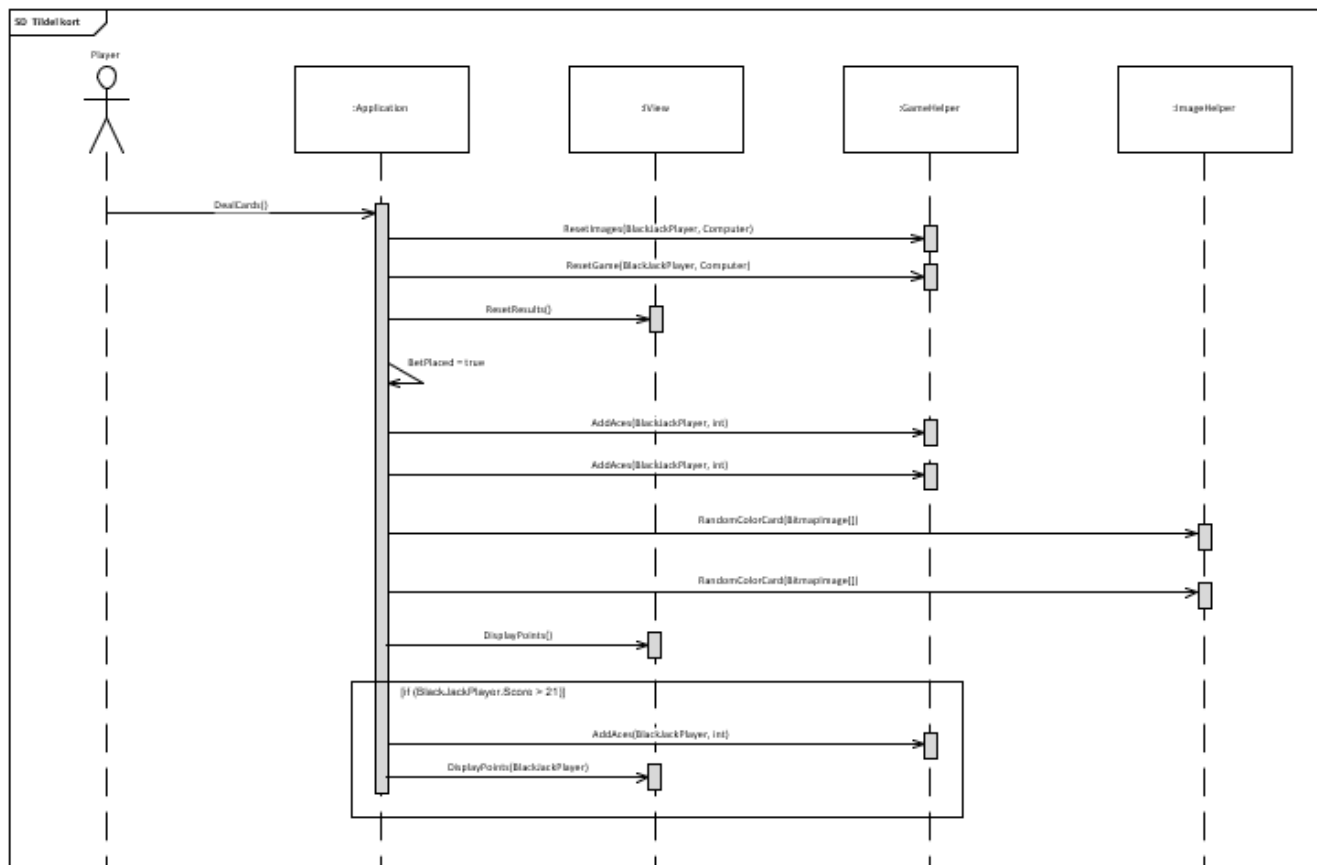
På figur 10 ses det fulde klassediagram for BlackJack med alle klasser og tilhørende attributter. Disse klasser er alle inkluderet i BlackJack-applikationen. Klassernes formål og anvendelse har gruppen skrevet i implementeringsafsnittet under BlackJack.



Figur 10: Klassediagram for hele BlackJack-applikationen

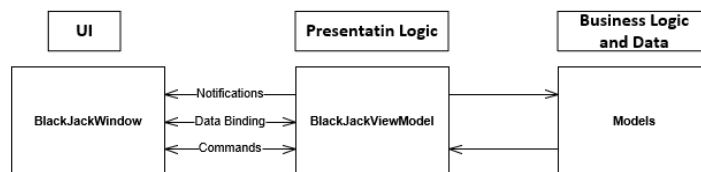
7.1.2 Sekvensdiagram

Denne tilhører klassediagrammet på figur 10. Sekvensdiagrammet her viser forløbet "Startspil". Startspil betegner det forløb, hvor brugeren allerede har åbnet BlackJack-applikationen og derefter starter et nyt spil. Det første der bliver kaldt, er metoder i GameHelper for at nulstille spillet ift. billeder og kort. Derefter nulstilles resultater i IView. Spilleren indtaster nu et gyldigt beløb, som han/hun ønsker at satse. Herefter tjekkes spillerens og dealerens (CPU) tildelte kort for essere. ImageHelper's metode RandomColorCard() giver kortene en tilfældig kulør, som på rigtige spillekort. Dealerens og spillerens point bliver vist gennem IView. Har man mere end 21 point, tjekker GameHelper om man har essere, hvorefter DisplayPoints() kaldes igen.



Figur 11: Sekvensdiagram for start af BlackJack.

Gruppen har haft stor gavn i, at anvende MVVM af flere årsager, men for at give den bedst mulige forklaring og overblik i teorien bag MVVM, vil gruppen se på et diagram, som gruppen har taget udgangspunkt i.



Figur 12: Diagram for MVVM

Diagrammet på figur 12 illustrerer det overordnede design for MVVM. Diagrammet viser, at gruppen har med tre separate klasser at arbejde med. Disse tre klassers interaktion med hinanden er essentielle for forståelsen af MVVM design pattern.

View er ansvarlig for alt det brugeren(end user) visuelt kan se og manipulere med. Der kan ud fra det udarbejdet diagram ses, at view har en relation til gruppens ViewModel, som holder på en tilstand af View. Diagrammet viser, at View er forbundet til ViewModel ved at anvende databinding og ved at sende kommandoer til ViewModel.

ViewModel har til formål at holde på View's tilstand og indeholder presentation logic for View. Det er ViewModels

opgave at fungere som et mellemled, der har til formål at forbinde View med Models. Dette er muligt for ViewModel at interagere med Model gennem properties, metodekald og kan modtage events fra Model klasserne.

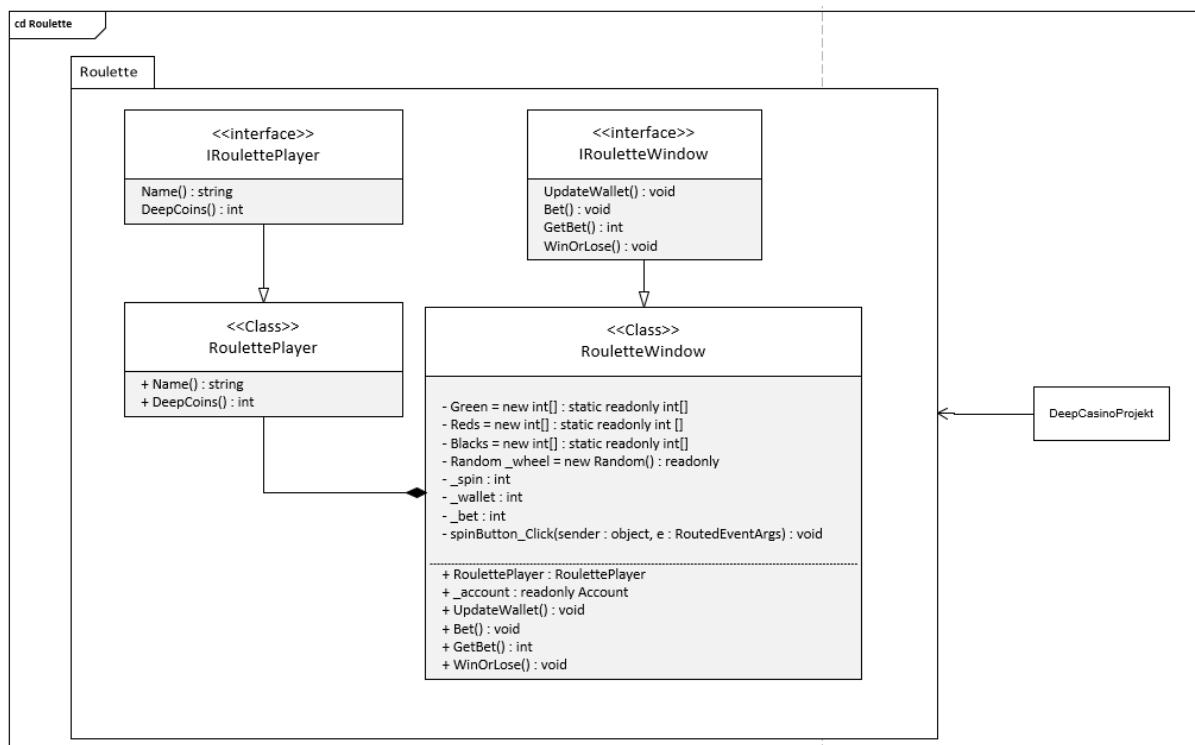
Model har til formål at holde på data og al business logic imens applikationen er under arbejde. I MVVM design har model ingen grund til at kende til hvilket GUI(Graphical User Interface) framework (fx WPF) der bliver anvendt, eller have kendskab til applikationsklasserne i præsentations laget dvs. View og ViewModel. Netop disse krav gør det nemt at genanvende Models i andre frameworks.

7.2 Roulette

I dette underafsnit behandles funktionaliteten af Roulette, hvor der fremgår et klassediagram med attributter, og et sekvensdiagram med funktionskald. Disse diagrammer er udarbejdet på baggrund af de hidtidige diagrammer fra afsnittet softwarearkitektur.

7.2.1 Klassediagram

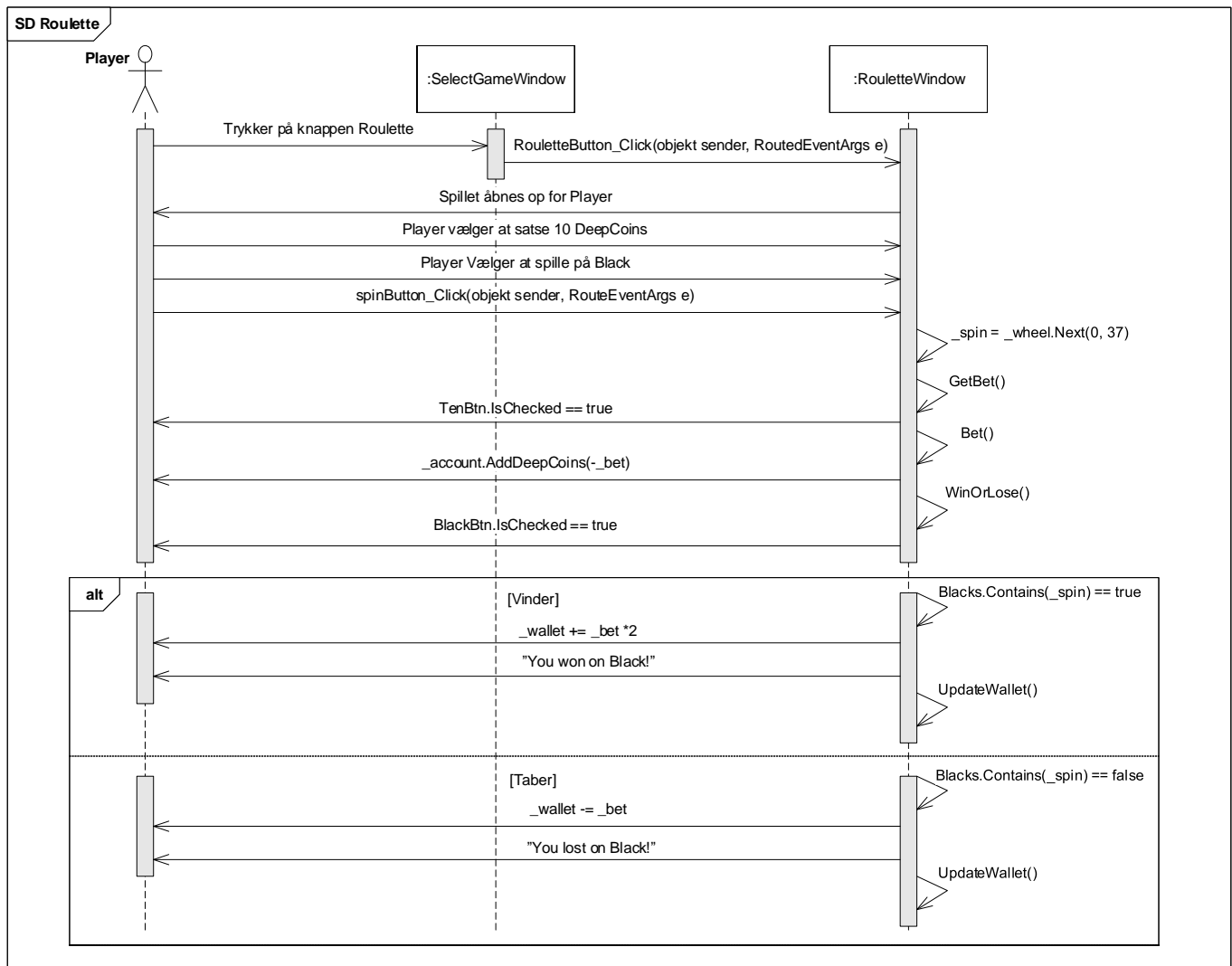
Her fremgår klassediagrammet til roulette, hvor members er indgået, som danner funktionaliteten der opfylder kravene til scenariet for sekvensdiagrammet. Figur 13 nedenfor viser overordnet kommunikationen mellem klasserne for både Roulette-applikationen og Opstarts-applikationen.



Figur 13: Klassediagram for Roulette-applikationen

7.2.2 Sekvensdiagram

Figur 14 nedenfor, illustrerer et af scenarierne for Roulette-applikationen, da de er ensartet. Her fremgår funktionaliteten i forhold til de fundne attributter for systemet. Scenariet omhandler at spilleren satser ti deepcoins på farven sort.



Figur 14: Sekvensdiagram for satsning på farven sort i Roulette-applikationen

Til start trykker Player på knappen Roulette, for at eksekvere Roulette-applikationen. Heraf vil `RouletteButton_Click()` funktionen køres, og applikationen åbnes. Det ses at Player vælger at satse ti deepcoins på farven sort. Dernæst trykker Player på Spin knappen, der eksekverer `spinButton_Click()`. `RouletteWindow`, generere et nummer mellem 0 til 37. Den vil dernæst kalde metoden `GetBet()`, som vil hente værdien for hvad Player har valgt at satse, efterfulgt kaldes metoden `Bet()`, som fratrækker de ti deepcoins fra Player's pung. På baggrund at Player har valgt at spille på hvad der ønskes, kaldes metoden `WinOrLose()`. Denne metode tjekker på hvad Player har valgt at spille på, samtidigt ville den også tjekke om Player vinder eller taber. Det ses, at hvis tilfældet lander på farven sort, vil Player vinde det dobbelte værdi af hvad Player har satset, og der udskrives tekststrengen "You won on Black!" på

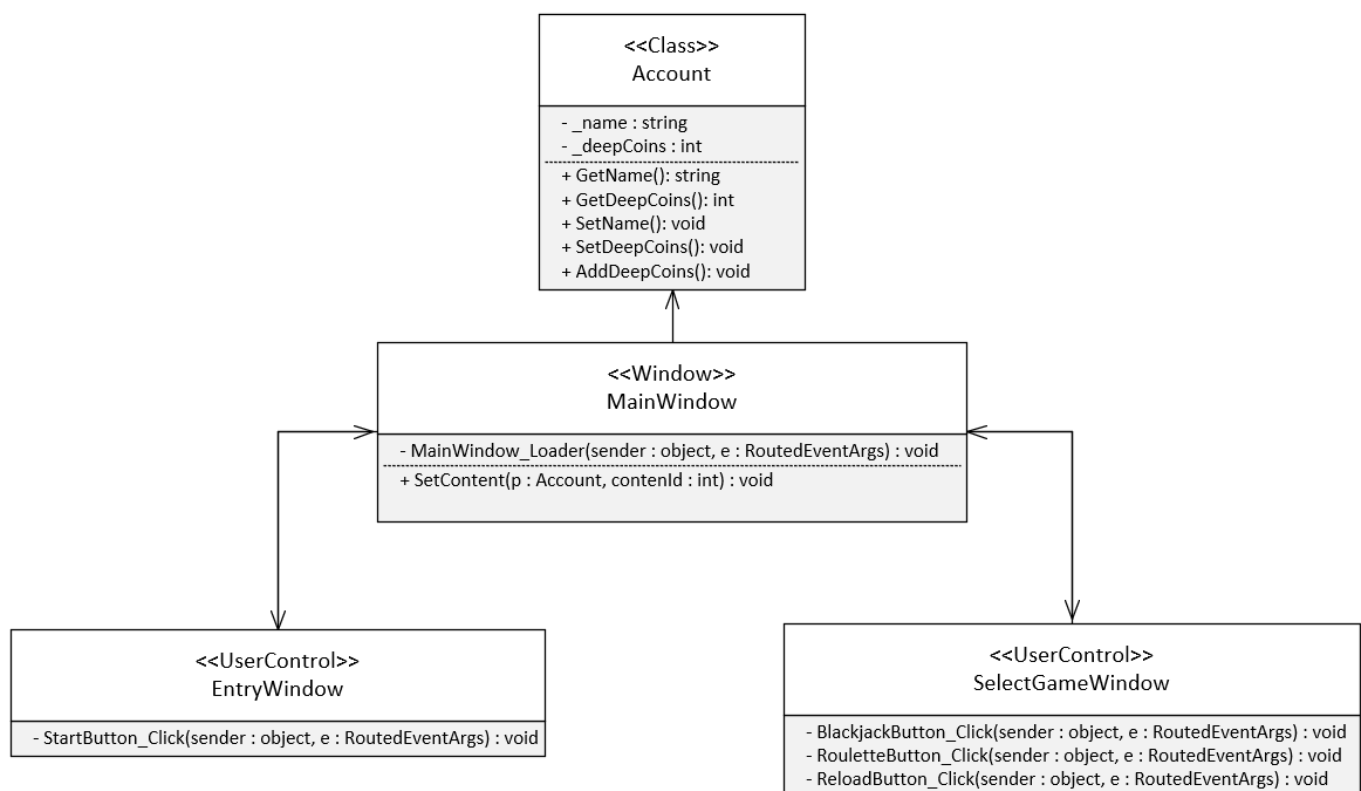
vinduet. Dernæst kaldes `UpdateWallet()` for at opdatere deepcoins tilhørende til Player. Hvis det er tilfældet at den ikke lander på farven sort, fratrækker den de ti deepcoins fra Player's pung, og der udskrives "You lost on Black!". Til sidst kaldes `UpdateWallet()` for at opdatere deepcoins tilhørende til Player.

7.3 Opstartsapplikation

Her beskrives den interne funktionalitet i de klasser som er beskrevet under "Opstartsapplikation" i afsnittet softwarearkitektur. Dette er grunden til at gruppen har inkluderet et mere detaljeret klasse- og sekvensdiagram med attributter og funktionskald i dette afsnit.

7.3.1 Klassediagram

På figur 15 ses et klassediagram over opstartsapplikationen. De forskellige blokke er beskrevet i arkitekturafsnittet, dog ses tilhørende attributter og metoder her.



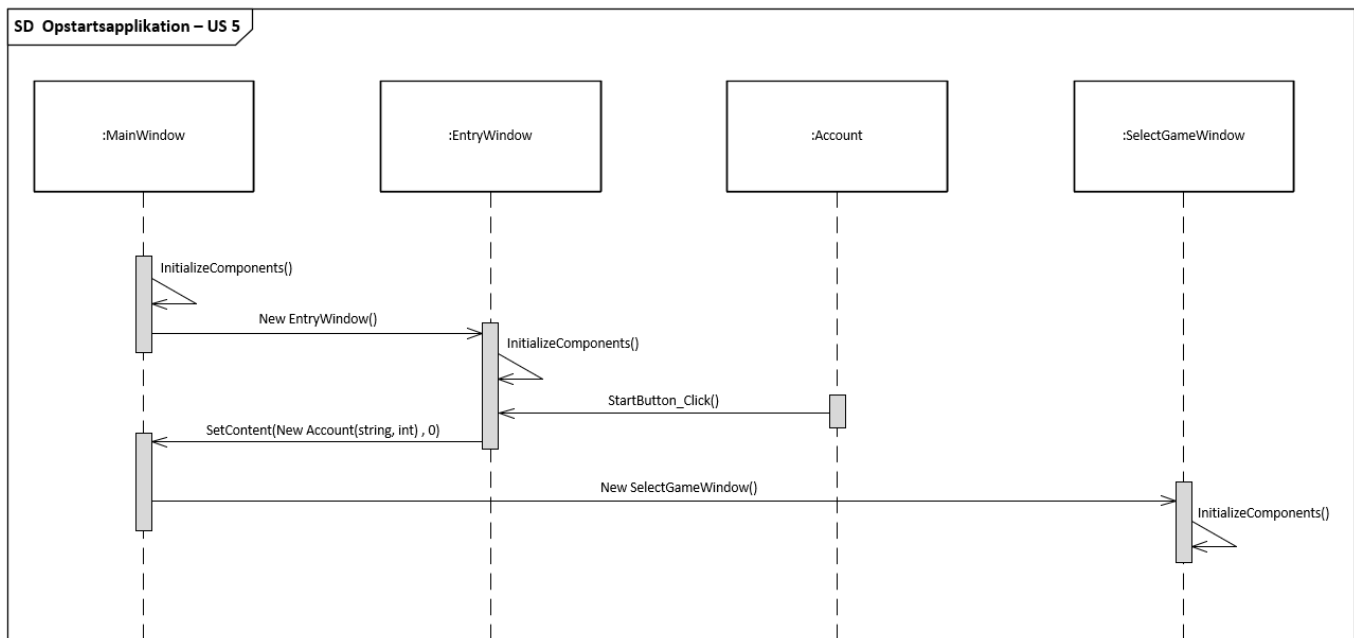
Figur 15: Klassediagram for opstartsapplikationen.

7.3.2 Sekvensdiagram

På sekvensdiagrammet (figur 16) illustreres hvornår metoderne fra klassediagrammet (figur 15) bliver kaldt og i hvilken rækkefølge. Yderligere beskrivelse af metodernes algoritme og funktion, finder sted i implementeringsdelen. En mindre gennemgang af trinene i diagrammet vil finde sted her. Bemærk, at så snart et vindue initieres kaldes funktionen `InitializeComponents()`, som findes inde i vinduernes (`MainWindow`, `EntryWindow`, `SelectGameWindow`)

constructors.

Det er MainWindow, hele programmet starter i, men så snart at den er initieret, oprettes et EntryWindow objekt. EntryWindow indeholder funktionen StartButton_Click, som venter på et brugerinput. Når den har fået sit brugerinput, får MainWindow besked og skifter vindue til SelectGameWindow.



Figur 16: Sekvensdiagram for opstartapplikationen.

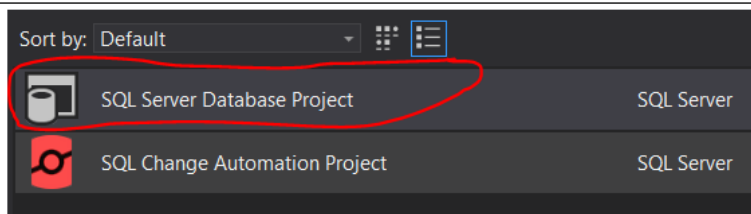
8 Software implementering

Under dette punkt vil der blive gennemgået, hvordan udarbejdningen af de forskellige arbejdsfordelinger af The Deep Casino er blevet implementeret.

8.1 Database

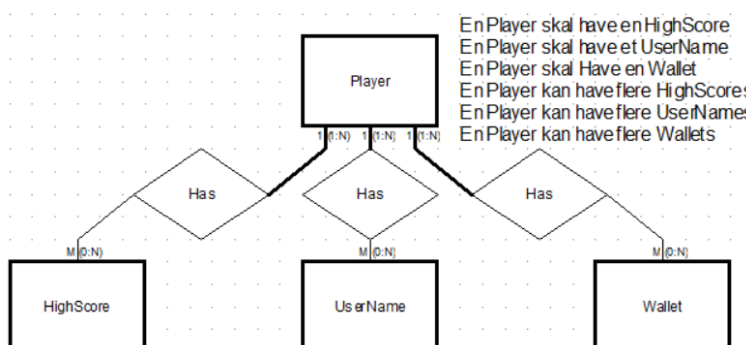
I forbindelse med udarbejdelsen af projektet er der blevet lavet en database der skal indeholde brugerens informationer såsom DeepCoins, Username osv. Den anvendte metode til at implementere databasen er en relationel database, hvori der også er implementeret CRUD-operationer. Den Relationelle database er arbejdet ud fra et logisk skema fra et program, DDS-Lite.

Det første man gør, er at oprette projektet, hvilket skal være en SQL Server Database Project, som kan ses forneden.



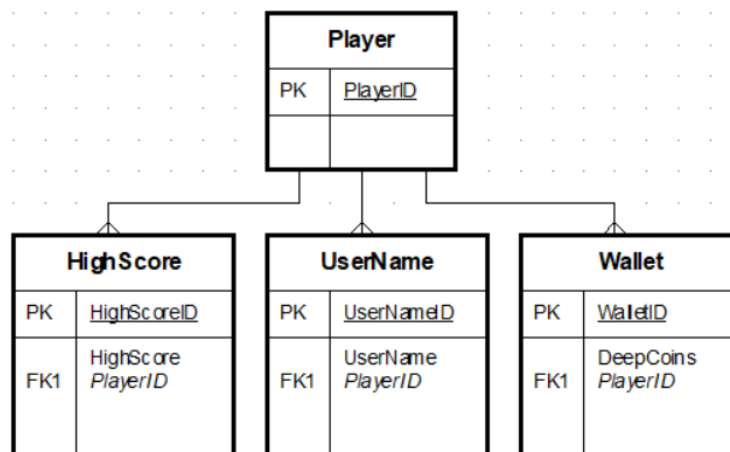
Figur 17: Oprettelse af SQL Server Database

DDS-Lite åbnes og der skal laves et ERD (Entity Relationship Diagram). Gruppens ERD for Databasen ses forneden:



Figur 18: Entity Relationship Diagram for Databasen

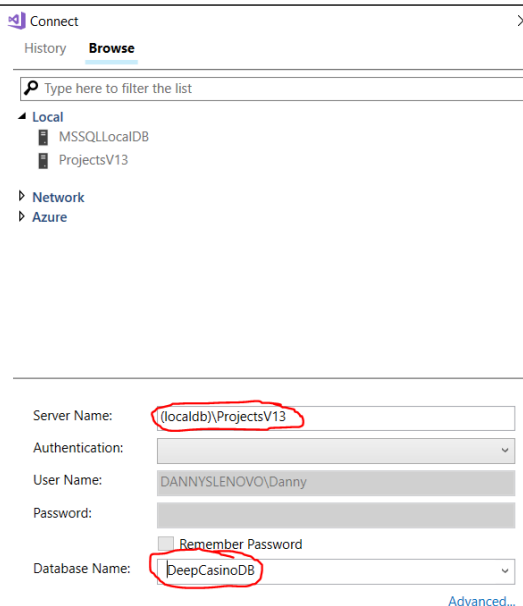
Ud fra ERD bliver der automatisk bygget en Crow's Foot diagram, som ses forneden.



Figur 19: Crow's Foot

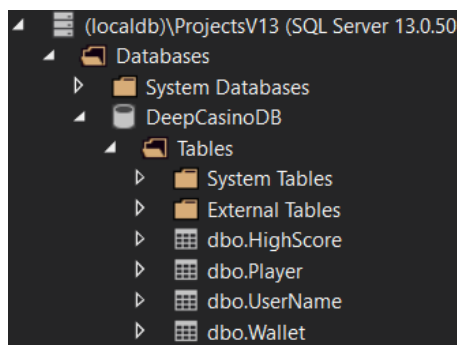
Som man kan se på Figur 19 kan man se hvilke properties hver entitet har og hvordan de er forbundet til hinanden. Inde på DDS-Lite kan man få bygget et logisk skema, som derefter kan sættes ind som en query inde på Visual Studio. Når man sætter det logiske skema ind i en query-fil, skal man eksekvere den, hvorefter den laver tabellerne i databasen.

Inden man eksekverer query-filen skal man sikre sig at det er den rigtige database den laver tabellerne på. Dette ses forneden markeret med rødt.



Figur 20: Forbindelse til Databasen via query-filen

Efter man har eksekveret query-filen får man nu tabellerne oprettet i databasen, uden nogle data, da man selv skal indsætte data vha. CRUD operationer. Tabellerne ses forneden.



Figur 21: De oprettede tabeller fra query-filen

8.1.1 CRUD operationer

Nu da tabellerne er på plads skal der nu laves CRUD operationer i en .cs fil. Der oprettes en cs-fil DeepCasinoDBUtil.cs, hvilket er her, CRUD-operationerne kan findes. På koden forneden ses hvordan den anvendte ConnectionString ser ud, når systemet kører på en server database. Den database der bliver anvendt er en database der er udgivet af vores underviser i E18I4DAB.

```

1 private SqlConnection OpenConnection
2 {
3     get
4     {
5         var con = new SqlConnection(@"Data Source=st-i4dab.uni.au.dk;Initial Catalog=
        E18I4DABau556770;User ID=E18I4DABau556770;Password=E18I4DABau556770;Connect Timeout=30;Encrypt

```

```

    =False; TrustServerCertificate=False; ApplicationIntent=ReadWrite; MultiSubnetFailover=False");
6
    con.Open();
7
    return con;
8
    }
9
10 }

```

Listing 1: ConnectionString

Som man kan se på koden ovenfor har gruppen angivet en connection string til en server database, som er den gule string. Det betyder, at de data der smides ind i tabellerne kommer til at være på server databasen.

Nu da gruppens ConnectionString er på plads, skal der nu implementeres CRUD-operationerne for de entiter der findes på gruppens ER-Diagram på Figur 18. Det første af CRUD-operationerne der bliver oprettet er Create, hvilket i dette tilfælde er "Add" som først tilføjes til entiteten, Player.

```

1 public void AddPlayerDB(ref Player pla)
2 {
3     string insertStringParam = @"INSERT INTO [Player] (Wallet, UserName, HighScore)
4                                     OUTPUT INSERTED.PlayerID
5                                     VALUES (@Wallet, @UserName, @HighScore)";
6
7     using (SqlCommand cmd = new SqlCommand(insertStringParam, OpenConnection))
8     {
9         cmd.Parameters.AddWithValue("@Wallet", pla.Wallet);
10        cmd.Parameters.AddWithValue("@UserName", pla.UserName);
11        cmd.Parameters.AddWithValue("@HighScore", pla.HighScore);
12
13        pla.PlayerID = (long)cmd.ExecuteScalar();
14    }
15 }

```

Listing 2: Databasens Create operation for Player

Der kan ses på koden ovenfor, at der for det første bliver oprettet en string der skal indeholde noget SQL sprog. Denne string bliver eksekveret når de nedenstående valideringer er korrekte. På linje 7 i koden kan man se, at vi bruger klassen SqlCommand, hvilket repræsenterer gemte procedurer til at eksekvere på en SQL server database. Der ses også, at der oprettes et SqlCommand objekt med to parametre, en string og en SqlConnection, hvilket betyder, at der initialiseres en instans af SqlCommand klassen. På linje 17 bliver der kørt en funktion `ExecuteScalar()` hvilket er under klassen, SqlCommand. Denne funktion modtager en enkel værdi fra databasen. Fra linje 9-11 kan man se, at der bruges `Parameters.AddWithValue(a,b)`, hvor `Parameters` er navnet på parametren der skal indsættes og `AddWithValue` bruges når der skal tilføjes en parameter med et specifikt navn og værdi.

Nu hvor Create operationen er på plads for Player skal der nu laves en Read operation for Player. Koden kan ses forneden.

```

1 public List<Player> GetPlayerDB()
2 {

```

```

3      string getStringParam = @"SELECT * FROM Player";
4      Console.WriteLine("PlayerID \t\t HighScore \t\t\t Username \t\t\t Wallet ");
5      using (var cmd = new SqlCommand(getStringParam, OpenConnection))
6      {
7          SqlDataReader rdr = null;
8          rdr = cmd.ExecuteReader();
9          List<Player> players = new List<Player>();
10         Player pla = null;
11         while (rdr.Read())
12         {
13             pla = new Player
14             {
15                 PlayerID = (long) rdr["PlayerID"],
16                 UserName = (string) rdr["UserName"],
17                 HighScore = (string) rdr["HighScore"],
18                 Wallet = (string) rdr["Wallet"]
19             };
20
21             Console.WriteLine(String.Format("{0} \t\t\t | {2} | \t\t\t | {1} | \t\t\t | {3} |",
22                 rdr[0], rdr[1], rdr[2], rdr[3]));
23
24             players.Add(pla);
25         }
26         return players;
27     }

```

Listing 3: Databasens Read operation for Player

Read operationen, som ses ovenfor, ligner meget Create, men denne her gang bliver der lavet en `List<Player>`, da der skal hentes en liste af de nuværende data'er i tabellen, Player. Man kan se på linje 8, at der bliver brugt `ExecuteReader()`, som der eksekverer kommandoer, der returnerer rækker. Den sender CommandText til Connection og bygger SqlDataReader ved at bruge en af CommandBehavior værdierne. Fra linje 15-18 tildeles attributterne i Player til de tilsvarende værdier i databasen, så man kan få udskrevet de rigtige værdier. Derefter bliver værdierne udskrevet på en terminal, som kan ses på linje 21. På linje 23 bliver Player objektet, pla, tilføjet til players og derefter bliver players returneret på linje 25.

Næste operation er Update, hvor man opdaterer de data der er blevet indsat i en given tabel. Koden for Update operationen kan ses forinden.

```

1 public void UpdatePlayerDB(ref Player pla)
2 {
3     string updateString = @"UPDATE Player
4         SET Wallet = @Wallet, Username = @Username, HighScore = @HighScore
5         WHERE PlayerID = @PlayerID";
6     using (SqlCommand cmd = new SqlCommand(updateString, OpenConnection))
7     {
8         cmd.Parameters.AddWithValue("@Wallet", pla.Wallet);

```

```

9      cmd.Parameters.AddWithValue( "@UserName" , pla.UserName);
10     cmd.Parameters.AddWithValue( "@HighScore" , pla.HighScore);
11     cmd.Parameters.AddWithValue( "@PlayerID" , pla.PlayerID);
12
13     var id = (long) cmd.ExecuteNonQuery();
14 }
15 }

```

Listing 4: Databasens Update operation for Player

Igen kan man se nogle sammenligninger med de andre operationer for Update operationen. Det foregår på samme måde som i Create-operationen. De eneste forskelle der er mellem de to operationer er SQL stringen og linje 13 på begge kode udsnit. På linje 13 ses, at der her i Update bliver kaldt en funktion, `ExecuteNonQuery()`. I Create operationen er der en `ExecuteScalar()`. `ExecuteNonQuery()` eksekverer kommandoer såsom INSERT, DELETE, UPDATE og SET statements.

Den sidste operation er Delete operationen, hvis formål er at slette noget data fra en given tabel. Implementeringen kan ses forneden.

```

1 public void DeletePlayerDB(ref Player pla)
2 {
3     string deleteString = @"DELETE FROM Player WHERE (PlayerID = @PlayerID)";
4     using (SqlCommand cmd = new SqlCommand(deleteString , OpenConnection))
5     {
6         cmd.Parameters.AddWithValue( "@PlayerID" , pla.PlayerID);
7
8         var id = (long) cmd.ExecuteNonQuery();
9         pla = null;
10    }
11 }

```

Listing 5: Databasens Delete operation for Player

Her står der ikke så markant meget som der gør i de andre operationer, da Delete operationens formål er at slette et given ID. I dette tilfælde er det en PlayerID der skal slettes. Som man kan se på SQL stringen på linje 3 sletter man fra Player tabellen. Når et ID slettes, slettes hele rækken.

Nu hvor CRUD-operationerne er på plads skal der nu laves en applikationen der kalder på disse operationer. Gruppen har valgt at kalde denne fil for DCAApp, forkortet DeepCasinoApp. Kaldene på CURD-operationerne kan ses forneden.

```

1 public class DCAApp
2 {
3     public void TheApp()
4     {
5         DeepCasinoDBUtil util = new DeepCasinoDBUtil();
6
7         Player newPlayer = new Player() { PlayerID = 0, UserName = "Navn", HighScore = "100",
8         Wallet = "Pung" };
9         util.AddPlayerDB(ref newPlayer);

```

```

9         util.DeletePlayerDB(ref newPlayer);
10        util.UpdatePlayerDB(ref newPlayer);
11        util.GetPlayerDB();
12    }
13 }

```

Listing 6: DCApp hvor der bliver kaldt på CRUD-operationerne

Som man kan se på Listing 6 bliver der for det første lavet et objekt af DeepCasinoUtil, som ses på linje 5, hvilket er den fil hvor CRUD-operationerne er i. Herefter bliver der lavet et objekt af Player med nogle værdier på linje 7. Fra linje 8-11 er der hvor vi kalder på de CRUD-operationer der blev oprettet. Man ser også, at der en reference til Player objektet med de specifikke værdier. **GetPlayerDB()** behøves ikke en reference da den skal udskrive det der er i databasen for Player. For at indsætte en Player, giver man Player objektet nogle værdier og kører koden. Det samme princip gælder hvis man ville opdatere eller slette en Player. Der skal bemærkes, at kodeudsnittet ovenfor kun gælder for Player og ikke de andre entiteter, såsom HighScore, UserName og Wallet.

Men koden kan ikke køre endnu uden et **Main()** program, så der bliver oprettet en ny .cs fil med et **Main()** program i. Koden for denne ses forneden.

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using ApplicationLogic;
7
8 namespace UserApplication
9 {
10     class Program
11     {
12         static void Main(string[] args)
13         {
14             DCApp runningApp = new DCApp();
15             runningApp.TheApp();
16         }
17     }
18 }

```

Listing 7: Main() program for Databasen

Som man kan se i kodeudsnittet ovenfor består vores **Main()** program af at kalde på vores DCApp. Der skal bemærkes, at CRUD-operationerne vist ovenfor er udelukkende for Player!

8.2 BlackJack

I dette afsnit har gruppen beskrevet de overvejelser, der har været under dannelsen samt implementeringen af den endelige solution mht. SOLID-principperne og MVVM.

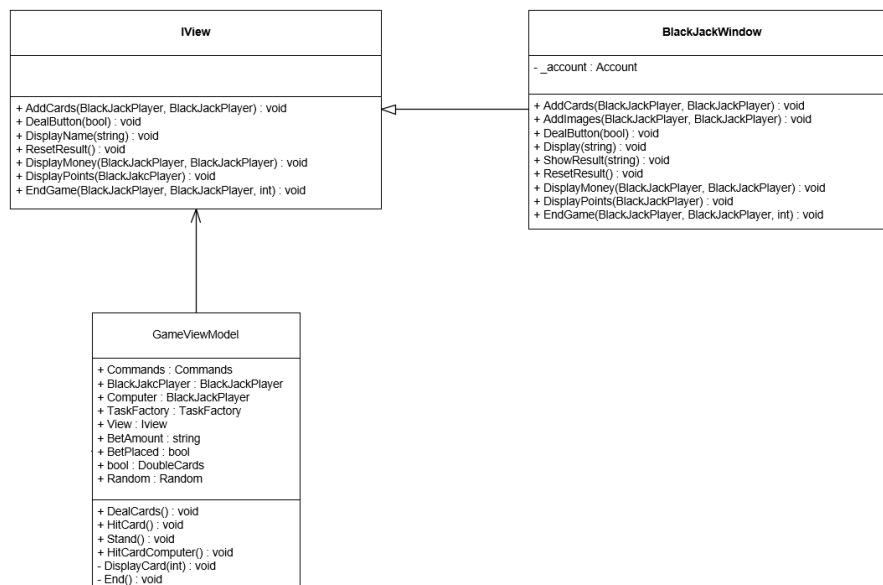
8.2.1 SOLID

S – Single responsibility princippet

Gruppens solution bære til dels præg af, at gruppen har haft SRP i baghovedet under udarbejdelsen af BlackJack-solutionen. Det ses specielt på klasserne inde under Commands-mappen. Gruppens DealCommand-klasse har eksempelvis kun ét ansvar, netop at sørge for at et kort bliver tildelt brugeren. Gruppen er opmærksomme på ikke at distribuere funktionalitet ud så meget at man får unødigt kompleksitet. Hvis en klasse såsom BlackJackPlayer ikke har behov for at ændre sig, er der ingen årsag til at bruge SRP.

D - Dependency inversion princippet

Gruppens BlackJack-solution benytter dette princip, da gruppen i koden har et interface, så det ikke er high-level modulet som opretter low-level modulet. Interfacet, som low-level modulet implementerer, sørger bl.a. for at underliggende funktionalitet let kan ændres fordi at high-level modulet nu ikke er forbundet direkte til den.



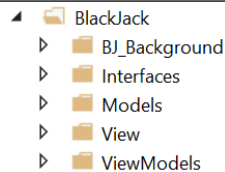
Figur 22: Diagram for anvendelse af DIP

På figur 12, som er et udsnit af gruppens klassediagram, illustreres hvordan gruppens high-level modul bruger interfacet IView som arves af BlackJackWindow.

8.2.2 MVVM - BlackJack

I dette afsnit vil gruppen gå i dybden med implementeringen MVVM samt de faktorer, der spiller en væsentlig rolle for at gøre det muligt at bruge design pattern i vores applikation. Ligeledes vil gruppen gøre brug af kode udsnit til, at illustrere hvordan data binding er anvendt i forbindelse med ICommand interfacet.

Gruppens fremgangsmåde for at gribe MVVM design pattern an på startede med at inddeling af de tre overordnet lag, som er beskrevet i analysen figur 12. Der ses her et skærbillede, som illustrerer dokument opdeling i gruppens Visual studio applikation.



Figur 23: Opstilling af BlackJack

Denne opstilling lægger op til brug af MVVM, og samtidigt gør det mere overskueligt for gruppens medlemmer når der bliver refereret til en bestemt klasse.

I View klassen finder vi BlackJackWindow.xaml samt BlackJackWindow.xaml.cs, der består af applikationens view, som har til formål at indkapsle userinterface. Det er denne del af applikationen, som brugeren interagerer med og visuelt kan se. Et ideelt eksempel af koden bag view vil ikke indeholde andet kode end en constructor, der kalder InitializeComponent-metoden. Det er dog ikke tilfældet for gruppens BlackJackWindow, eftersom vi har implementeret metoder som udfører visuelle adfærd. Dette skyldes primært at det har været vanskeligt eller uhensigtsmæssigt at implementere i XAML(Extensible Application Markup Language) koden. Et eksempel på dette vil være vores addImage, som har til formål at visuelt tilføje et kort når brugeren har brug for det:

```

1  private void AddImages(BlackJackPlayer player , Grid grid)
2      {
3          for (var i = 0; i < VisualTreeHelper.ChildrenCount(grid); i++)
4          {
5              var child = VisualTreeHelper.GetChild(grid , i);
6              player.Images.Add((Image)child);
7          }
8      }

```

Listing 8: AddImages

Ovenstående eksempel gør gruppen brug af funktionen AddImages, der har til formål at tilføje billeder i en flydende overgang ved at anvende VisualTreeHelper, der har til formål at hente kort billede alt afhængigt af billede.

Der ses på diagram 12, at BlackJackWindow skal have kendskab til BlackJackViewModel, hvilket bliver gjort i constructoren for det bagvedliggende kode for BlackJackWindow. Fremgangsmåden er at sætte DataContext til at være lig med BlackJackViewModel. Dette gør at vi nu har kontrollen til BlackjackViewModel.

BlackJackViewModel har til formål at indkapsle presentation logic og data for BlackJackWindow. Den har ikke nogen direkte reference til BlackJackWindow eller kendskab til BlackJackWindow specifikke implementation eller type.

Gruppens BlackJackWindow interagerer med BlackJackViewModel gennem databinding, kommandoer og ændringer af events via notifikationer. BlackJackViewModel implementer proprietæren commands, der holder på en reference af alle vores øvrige commands(Hit, Stand og Deal). Disse commands er forbundet til view via databinding og commands i vores XAML kode.

```
1 <Button Command="{ Binding Commands.HitCommand }" Name="Hit" Content="Hit!" />
2 <Button Command="{ Binding Commands.StandCommand }" Name="Stand" Content="Stand!" />
3 <Button Command="{ Binding Commands.DealCommand }" Name="Deal" Content="Deal!" />
```

Listing 9: AddImages

Det ses oftest at disse commands bliver behandlet i viewmodel(BlackJackViewModel), men gruppen har for overskuelighedens skyldt valgt at implementer dem i hver deres klasse i hhv, Deal, Hit og StandCommand.

Commands er en implementation af ICommand interfacet, der er en del af det anvendte .NET Framework. Dette interface er meget velkendt i mange MVVM applikationer, hvilket skyldes de tre members ICommand interfacet specificer. Disse tre member funktioner er årsagen til den løse kobling og gruppen ser det derfor relevant at gå i dybden med dem. Gruppen vil tag udgangspunkt i DealCommand:

```
1 public void Execute(object parameter)
2 {
3     _viewModel.DealCards();
4 }
```

Listing 10: Execute

Metoden Execute kaldes, når kommandoen i XAML koden aktiveres. Den holder på en parameter(object), som kan bruges til at videregive yderligere oplysninger fra caller til kommandoen. I listing 10 ser vi XAML koden at Commands er blevet til DealCommand, og det er når denne knap bliver trykket på af brugeren at Execute vil blive kaldt, og dermed kalde funktionen DealCards, der har til formål at uddele kort til Dealer og brugeren.

```
1
2 public bool CanExecute(object parameter)
3 {
4     if (_viewModel.BetPlaced)
5     {
6         return false;
7     }
8     bool parsed = int.TryParse(_viewModel.BetAmount, out var bet);
9     if (parsed)
10    {
11        if (bet >= 1 && bet <= 500)
12        {
13            return true;
14        }
15    }
16    return false;
17 }
```

Listing 11: CanExecute

Denne metode har til formål at beslutte hvorvidt kommandoen kan eksekveres i dens pågældende tilstand. Dette ses i vores kode eksempel listing 12, at dette kun er muligt hvis brugeren har sat et beløb ind, som han ønsker at

spille på. Funktionen CanExecute har parameteren object, der holder på data som bliver brugt af kommandoen. CanExecute returner boolsk værdi true eller false altafhængigt af om brugeren har indtastet den rigtige værdi dvs. CanExecute bliver ikke eksekveret hvis brugeren indtaster et beløb som er mindre end 1 eller større end 500.

Gruppen illustrerer her metoden CanExecuteChanged:

```
1 public event EventHandler CanExecuteChanged
2 {
3     add => CommandManager.RequerySuggested += value;
4     remove => CommandManager.RequerySuggested -= value;
5 }
```

Listing 12: CanExecuteChanged

Dette event aktiveres af kommandoen for at notificere sine Consumers (eksempelvis gruppens knapper) om at CanExecute-metoden muligvis har ændret sig. I XAML, når en forekomst af ICommand er bundet til en kommandos kommando-property gennem en dataindbinding, vil CanExecuteChanged-eventet automatisk kalde CanExecute-metoden, og control vil blive aktiveret eller deaktiveret alt afhængigt af den ønskede funktionalitet.

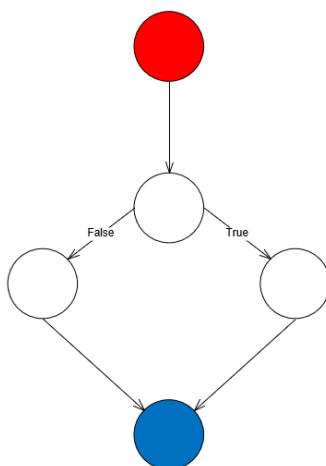
ICommand er blevet anvendt flittigt af gruppen og gjort det muligt for os at anvende MVVM design pattern. Ulempen ved vores fremgangsmåde er dog, at hvis vi ønsker at udvide vores applikation til at have flere kommandoer så er det upraktisk at implementere ICommand for hver eneste kommando. Det vil her være nyttigt at gøre brug af det populære framework relaycommand, der har en generisk implantation af ICommand.

8.2.3 Statisk analyse og Software matrice

Gruppen vil i dette afsnit give et eksempel på hvordan Cyclomatic Complexity er udregnet med udgangspunkt i en simpel funktion. Eksempel på hvordan man kan udregne Cyclomatic Complexity:

```
1 public void DisplayPoints(BlackJackPlayer player)
2 {
3     if (player.Name == "Computer")
4     {
5         ComputerPoints.Content = player.Score;
6     }
7     else
8     {
9         PlayerPoints.Content = player.Score;
10    }
11 }
```

Listing 13: Kode



Figur 24: Flow chart of DisplayPoints

Knuderne i flowchart diagrammet repræsenterer kode der eksekveres, mens kanter repræsenterer kontrolflow mellem knuderne. Udefra diagrammet og i forlængelse af vores kode kan det ses at vi har fire knuder og fem kanter. Dette giver os formelen:

$$2 = 5 - 5 + 2 * 1$$

Dette fortæller os at vi skal have lavt to teste cases for at få fuld covarge på denne funktion. Det udregnet resultatet af Cyclomatic Complexity stemmer ligeledes i overensstemmelse med listing(kode) [ref]. Koden er bestående af en if/else statement, hvilket vil sige at der kan være to resultat. Det vil her være oplagt at anvende unittest for at reelt kunne dokumenterer udkommet af koden.

8.3 Roulette

I dette afsnit vil implementeringen af Roulette-applikationen behandles. Her fremgår en gennemgang af det interne funktionalitet af applikationen. Der vil være indgået kodeudsnit og beskrivelser på de mest betydende metoder.

Det skal nævnes, at der i det senere afsnit, Diskussion, er fortalt at der ikke er anvendt af et design pattern.

På baggrund af det valgte scenarie for sekvensdiagrammet fra afsnittet Softwaredesign, er det valgt at fokusere på metoderne der er anvendt derpå.

8.3.1 Private properties

Disse private objekter er essentielle for applikationen, da de afspejler spille-reglerne og essensen af spillet, roulette. På det nedenstående kodeudsnit, ses de private objekter.

```

1  private static readonly int[] Green =
2  {

```

```

3      0
4  };
5
6  private static readonly int[] Reds =
7  {
8      32, 19, 21, 25, 34, 27, 36, 30, 23, 5, 16, 1, 14, 9, 18, 7, 12, 3
9  };
10
11 private static readonly int[] Blacks =
12 {
13     15, 4, 2, 17, 6, 13, 11, 8, 10, 24, 33, 20, 31, 22, 29, 28, 35, 26
14 };
15
16 private readonly Random _wheel = new Random();
17 private int _spin;
18 private int _wallet;
19 private int _bet;

```

Listing 14: Private properties i RouletteWindow

På det øvrige kodeudsnit, Private properties, ses tre integer arrays; Green, Reds og Blacks. Disse tre integer arrays afspejler hjulet i spillet roulette. Her indgår farverne grøn, rød og sort. Hver af de arrays indeholder en række tal, som kendetegner hvilke tal der indebærer farven. Endvidere er der den oprettede objekt, `_wheel`, som er det der anvendes til at generere et nummer mellem 0 til 37. Dernæst er der `_spin`, som har til formål at indeholde det genereret nummer. Yderligere et objekt kaldet `_wallet`, som har til formål at indeholde deepcoins tilhørende til Player. Til sidst objektet `_bet`, der har til formål at indeholde værdien Player gerne vil satse.

8.3.2 spinButton_Click()

Denne funktion har til formål at spinne hjulet og generere et nummer fra 0 til 37. Den er bundet til knappen Spin, der visuelt ses på applikationen. Hovedsageligt er denne funktion start-knappen til hele applikationen. Kodeudsnittet ses nedenfor.

```

1  private void spinButton_Click(object sender, RoutedEventArgs e)
2  {
3      _spin = _wheel.Next(0, 37);
4      SpinLabel.Content = "You landed " + _spin;
5
6      EmptyWalletLabel.Content = "";
7      WinOrLose();
8      UpdateWallet();
9  }

```

Listing 15: spinButton_Click()

Det ses på det ovenstående kodeudsnit, at funktionen starter med at generere et tilfældigt nummer fra 0 til 37. Dernæst udskriver den hvilket nummer der er blevet genereret og kalder funktionen `WinOrLose()`. Dette vil køre

sekvensen for om man vinder eller taber, og derfra udskrive resultatet på det genereret nummer. Dernæst kaldes funktionen `UpdateWallet()`, som anvendes for at sikre at pungen viser det oprindelige værdi i applikationen.

8.3.3 UpdateWallet()

Denne funktion har til formål at opdatere pungen for Player. Det ses på kodeudsnittet nedenfor, at den anvender sig af det private objekt `_wallet`. Hvis værdien er under 0, skal det give en advarsel til Player, at vedkommende ikke har flere deepcoins på kontoen, derved udskrives en tekststreng, der siger "You do not have enough DeepCoins!". Hvis værdien er større end 0, vises den nuværende værdi i `_wallet`.

```
1  public void UpdateWallet()
2  {
3      if (_wallet < 0)
4      {
5          EmptyWalletLabel.Content = "You do not have enough DeepCoins!";
6      }
7      else
8          WalletLabel.Content = "DeepCoins: " + _wallet;
9
10 }
```

Listing 16: UpdateWallet()

8.3.4 GetBet()

`GetBet()` funktionen har til formål at returnere den værdi, som Player ønsker at satse. Her ses at Player har mulighed for at satse henholdsvis 5, 10, 50 og 100 deepcoins hver runde.

```
1  public int GetBet()
2  {
3      if (FiveBtn.IsChecked == true) return 5;
4      if (TenBtn.IsChecked == true) return 10;
5      if (FiftyBtn.IsChecked == true) return 50;
6      if (HundredBtn.IsChecked == true) return 100;
7      return 0;
8  }
```

Listing 17: GetBet()

I det senere underafsnit forklares brugen af `GetBet()`.

8.3.5 Bet()

`Bet()` funktionen beskæftiger sig med satsningen for applikationen. Funktionen anvender sig af `_bet`, som kan ses ovenstående kodeudsnit, Private properties. Samt anvender den sig også af `GetBet()` funktionen. Det ses på kodeudsnittet nedenfor, at den først tilhænger `_bet` til en værdi som er returneret fra `GetBet()`. Derfra går den i en if-else statement, som først sikre om det er muligt at satse den spørgende værdi fra Player. Hvis det er tilfældet at

den siger OK til hvad `_bet` indeholder, vil den fratrække det fra Player's pung, hvorefter `UpdateWallet()` kaldes for at opdatere `_wallet`.

```
1 public void Bet ()
2 {
3     _bet = GetBet ();
4     if (_bet <= _account.GetDeepCoins())
5     {
6         _account.AddDeepCoins(-_bet);
7         UpdateWallet();
8     }
9 }
```

Listing 18: Bet()

8.3.6 WinOrLose()

Denne funktion består i bund og grund kun af if-else statements. Den anvender sig af `_spin` og `_bet` objekter, samt `Bet()` og `UpdateWallet()` funktionerne. Den har til formål, at fortælle om Player vinder eller taber på sit spil. Et godt indblik på sekvensen af denne funktion kan ses i afsnittet Softwaredesign for Roulette.

Det ses at alle if-else statements er ensartet. Til start vil funktionen kalde `Bet()`, som vil tilhænge værdien, Player har valgt at satse. Afhængig af hvad Player har valgt at satse på, vil det blive behandlet af `WinOrLose()`. Der tjekkes på lige/ulige/rød/sort/grøn, alt efter hvad der er valgt, vil den udføre if-else statement tilhørende til det. Heraf kommer der en "condition", som tjekker om Player vinder eller taber. Eksempelvis, hvis Player vælger at spille på lige-tal, så er conditionen, at hvis `(_spin % 2 == 0)`, så er conditionen *true*. Dette vil betyde at Player har vundet. Hvis det ikke er tilfældet, taber Player.

De resterende if-else statements foregår på samme måde, men hver især har en anden condition.

```
1 public void WinOrLose()
2 {
3     Bet();
4     if (EvenBtn.IsChecked == true)
5     {
6         if (_spin % 2 == 0)
7         {
8             ResultLabel.Content = "You won on Even Numbers!";
9             _wallet += _bet * 2;
10        }
11        else
12        {
13            ResultLabel.Content = "You lose on Even Numbers!";
14            _wallet -= _bet;
15        }
16    }
17 }
```



```
18     if (OddBtn.IsChecked == true)
19     {
20         if (_spin % 2 != 0)
21         {
22             ResultLabel.Content = "You won on Odd Numbers!";
23             _wallet += _bet * 2;
24         }
25         else
26         {
27             ResultLabel.Content = "You lost on Odd Numbers!";
28             _wallet -= _bet;
29         }
30     }
31
32     if (BlackBtn.IsChecked == true)
33     {
34         if (Blacks.Contains(_spin))
35         {
36             ResultLabel.Content = "You won on Black!";
37             _wallet += _bet * 2;
38         }
39         else
40         {
41             ResultLabel.Content = "You lost on Black!";
42             _wallet -= _bet;
43         }
44     }
45
46     if (RedBtn.IsChecked == true)
47     {
48         if (Reds.Contains(_spin))
49         {
50             ResultLabel.Content = "You won on Red!";
51             _wallet += _bet * 2;
52         }
53         else
54         {
55             ResultLabel.Content = "You lost on Red!";
56             _wallet -= _bet;
57         }
58     }
59
60     if (GreenBtn.IsChecked == true)
61     {
62         if (Green.Contains(_spin))
63         {
64             ResultLabel.Content = "You won on Green!";
65             _wallet += _bet * 36;
```

```
66         }
67         else
68         {
69             ResultLabel.Content = "You lost on Green!";
70             _wallet -= _bet * 18;
71         }
72     }
73     UpdateWallet();
74 }
```

Listing 19: WinOrLose()

8.4 Opstartsapplikation

Her kommer gruppen ind på opstartsapplikationen, hvor der fremgår en gennemgang af metoderne i de brugte klasser og vinduer. Med User story 5 i baghovedet beskriver gruppen med kodeudsnit de vigtigste metoder. I foregående afsnit(Softwaredesign) er disse metoder nævnt i forbindelse med et klasse- og sekvensdiagram.

Den mest betydende funktion i gruppens MainWindow.cs er metoden SetContent(Account, int):

```
1     public void SetContent(Account p, int contentId)
2     {
3         switch (contentId)
4         {
5             case 0: ContentHolder.Content = new SelectGameWindow(p); break;
6             case 1: ContentHolder.Content = new BlackJackWindow(p); break;
7             case 2: ContentHolder.Content = new RouletteWindow(p); break;
8         }
9     }
```

Listing 20: SetContent(Account, int)

Denne funktion har til formål at gøre det muligt at skifte mellem vinduer, så der ikke skal åbnes et nyt vindue ved åbning af eksempelvis RouletteWindow. Derudover vil den også videregive Account til de forskellige vinduer, som findes i applikationen. Derved beholder brugeren sine informationer på tværs af spillene. Som parametre tager funktionen derfor en Account og en integer. Denne integer sættes ind i en switch-case, som bestemmer hvilket vindue som applikationen skal skiftes til alt efter om den får et 0-, 1- eller 2-tal.

Bemærk, at ContentHolder er navnet på gruppens ContentControl. ContentHolder.Content bliver sat til et nyt EntryWindow i MainWindow's constructor. Derved ses EntryWindow som det første når applikationen køre.

```
1     public MainWindow()
2     {
3         InitializeComponent();
4         ContentHolder.Content = new EntryWindow();
5     }
```

Listing 21: MainWindow's constructor

Følgende metode hører hjemme i klassen `SelectGameWindow`.

```
1     private void BlackjackButton_Click(object sender, RoutedEventArgs e)
2     {
3         MainWindow window = (MainWindow)Window.GetWindow(this);
4         window.SetContent(Account, 1);
5     }
```

Listing 22: Metoden `BlackjackButton_Click`

Denne metode kaldes så snart at brugeren fra `SelectGameWindow` trykker på knappen "BlackJack". Det mest bemærkelsesværdige er at `contentId` bliver sat til 1, og på den måde skifter vinduet til `BlackJackWindow`.

De øvrige knapper i de forskellige vinduer, fungerer på samme måde. Princippet er at `contentId` skiftes alt efter knappens funktion. Derfor finder gruppen det ikke nødvendigt at beskrive flere funktioner herunder dette afsnit.

9 Test

Herunder ville der blive beskrevet hvorledes man har tænkt sig at teste de forskellige funktionaliteter af DeepCasino, samt hvad det forventede resultat er af disse tests.

9.1 Database - Test

Under dette punkt ville der blive lavet unit tests af Databasen for at se om de forskellige funktionaliteter virker efter hensigten. Det forventes, at de implementerede operationer, som er et krav for vores database, virker efter hensigten. Databasen har fire grundlæggende operationer, som har hver sine metoder til at udføre en bestemt handling. Disse operationer har til formål at kunne oprette en ny player i databasen med en række af informationer, som viser, hvad player's brugernavn er, hvad player's score er, og hvor meget optjent penge, player har til rådighed. De operationer, der skal testes, kaldes CRUD-operationer. Disse operationer bliver gennemgået i de efterfølgende punkter.

9.1.1 CRUD operationer - Test

Under dette punkt vil der udelukkende testes for CRUD-operationerne for Player. Der er blevet implementeret fire CRUD-operationer, der har hver deres metoder. CRUD-operationerne er følgende: Create, Read, Update og Delete. Den første CRUD-operation, der bliver testet er Create, som har til formål at oprette en ny player i databasen med tilhørende værdier.

9.1.1.1 Create - Test

Der testes nu om Create operationen virker efter hensigten. Det første der skal ske er, at man ændrer i DCAApp.cs filen. Man angiver derefter nogle værdier man vil have smidt ind i tabellen for Player. Man skal huske at angive PlayerID som 0, da den selv angiver PlayerID, alt efter om der allerede er noget data inde i tabellen for Player. Man skal også huske at udkommentere de andre operationer ellers vil alle CRUD-operationer køre på samme tid, hvilket ikke er hensigten. Forudsætningen for, at Create operationen skal virke er, at der skal være en database tilgængelig. Hvis ikke der er en database tilgængelig vil der opstå compilerfejl, eftersom dataen man prøver at indsætte ikke bliver indsat nogle steder henne. Efter man har angivet nogle specifikke værdier for Player, husker man at sætte UserApplication som Startup projekt, da UserApplication indeholder Main programmet. Man eksekverer derefter programmet og tjekker så om tabellen, Player, har fået nogle værdier indsat. For at tjekke om der indsat noget data i Player, skal man over i SQL Server Object Explorer og finde databasen der indeholder Player, højre-klikke og trykke "View Data", hvilket er herinde man finder tabellen for Player.

9.1.1.2 Read - Test

Nu hvor Create operationen er testet, skal der nu testes for Read operationen. Forudsætningen for, at Read operationen skal virke optimalt er, at Create operationen virker og, at der allerede er indsat nogle data på forhånd, eftersom man skal kunne udskrive tabellen i et terminalvindue. Den virker allerede hvis der ikke er indsat nogle

data, men hvis den skal udskrive noget data skal der på forhånd være indsat nogle data. Samme princip gælder, som i Create operationen, når man skal teste Read operationen. Man ændrer i DCAApp.cs filen og udkommenterer alle andre operationer udover Read operationen. Så eksekverer man programmet, hvorefter tabellen for Player vil blive udskrevet på et terminalvindue. Man ændrer ikke nogle værdier for Player, da Read operationens formål er at udskrive tabellen.

9.1.1.3 Update - Test

Operationen Update skal testes under dette punkt. Det er en forudsætning, at en player oprettet, før Update-operationen kan fungere, da den går ind og opdatere en player's værdier. Fremgangsmåden for Update-operationen er, at der skal være kendskab til det ID, som en player er oprettet med. ID'et er en nødvendighed, da det skal specificeres, hvilket player der skal have dens værdier opdateret. De værdier, som kan opdateres, er Wallet, UserName og HighScore. I DCAApp.cs-filen sker al oprettelse, opdatering og fjernelse af player. Det er i denne fil (DCAApp.cs), hvor en ny player oprettes med dens værdier angivet i parametrene. For at opdatere en player, skal playerens ID angives som parameter, hvorefter de ønskede ændringer angives i de andre felter som parametre. Herefter bliver metoden UpdatePlayerDB kaldt, som ligger i filen DeepCasinoDBUtil.cs. Denne metode sørger for at få indsat de nye parametre til den ønskede player. Når en player er opdateret som beskrevet foroven, tjekkes der efter i tabellerne i databasen, om de nye værdier stemmer overens med de nye værdier, som blev indtastet.

9.1.1.4 Delete - Test

Under dette punkt skal operationen Delete testes. For at teste denne operation, er det en forudsætning, at en player er oprettet i databasen. Delete-operationen har det formål, at den går ind og sletter en player fra databasen med dens tilhørende værdier. Fremgangsmetoden for Delete-operationen er, at der skal være kendskab til den player, hvilket ønskes slettet. ID'et er nødvendigt at have kendskab til, da det skal specificeres hvilken player, der ønskes slettet fra databasen. De tilhørende værdier for en player er: Wallet, UserName og HighScore. I filen DCAApp.cs bliver en ny player oprettet med tilhørende værdier. For at slette en player fra databasen skal den ønskede player's ID angives som parameter. Metoden DeletePlayerDB bliver kaldt, hvilket ligger i DeepCasinoDBUtil.cs, som sørger for at fjerne den ønskede player fra databasen. Når en player er slettet, tjekkes det efter i tabellerne i databasen, om den ønskede player er slettet fra databasen.

9.1.2 Forbindelsesledet mellem databasen og DeepCasino - Test

Der testes forbindelse mellem databasen og DeepCasino ved at der sker et event, når man opretter en bruger på applikationen, hvor der vil overføre nogle data til databasen. For at teste at dataene er blevet overført, så har man startet med at kører applikationen og dermed opretter en bruger ved at indtaste et navn. Efter man har indtastet et navn på Textbox og trykker på knappen, så i det øjeblik vil der ske et event. Der bliver oprettet en SQL-forbindelse til det lokale-Database ((localdb)), hvor brugerens navn og DeepCoins bliver gemt i databasen. Når man opdaterer tabellen i databasen, vil man være i stand til at kunne se brugerens navn og DeepCoins. Efter dataene er blevet overført til det lokale-database, så skal det testet med at oprette en SQL-forbindelse til

en SQL-server(st-i4dab.uni.au.dk). Når man oprette en forbindelse til denne server, så vil man være i stand til at kunne modtage dette data fra en anden enhed. Denne server er så blevet testet på den samme måde som den lokale-database.

9.2 BlackJack - Test

I dette afsnit kommer gruppen ind på, hvilke af BlackJack-applikationens funktionalitet brugeren har mulighed for at udføre. Dette vil sige, at gruppen gennemgår de moduler, som gruppen har kunne teste og hvorledes testene foregik. Yderligere er der en beskrivelse af hvorledes gruppen har båret sig an med integrations- og accepttest. Resultat af de enkelte test kan ses under resultat afsnittet.

Her adresseres User Story 3, som lyder på at "Brugeren skal kunne spille BlackJack" og User Story 1, som lyder på at "Brugeren skal kunne indtjene DeepCoins ved at vinde spil". Da der findes mange variationer af spillet BlackJack, vil der derfor blive refereret til ordlisten, som kan findes i dokumentationen. Det første der ønskes testet for applikationen er, at BlackJack-spilleren er korrekt indstillet med navn og DeepCoins, og at det kommer fra opstartsapplikationen, hvor brugeren er gemt i Account. Gruppen har efterfølgende også testet vha. debug om brugerens DeepCoins inkrementeres eller dekrementeres al afhængigt om brugeren vinder eller taber. Dette blev testet i klassen BlackJackWindow-filen.

Udover de nævnte test er BlackJack-kommandoer "Deal", "Hit" og "Stand" ligeledes blevet testet med samme fremgangsmåde. Gruppen har visuelt kunne se, at funktionaliteten af hver enkelt kommando er funktionel. Dette er ligeledes dokumenteret i resultatafsnittet, hvor det ses kort bliver udelt til brugeren samt dealer idet der bliver trykket på "Deal" knappen. Yderligere ses det at brugeren både har mulighed for at "Stand" og "Hit", men her skal der bemærkes at disse først er tilgængeligt efter brugeren har trykket på "Deal", hvilket virker hensigten.

Da Unit Test ikke er blevet implementeret, har Gruppen beskrevet eksempler, hvor der er kørt debugging på den udarbejdede kode. Debug gør der her muligt, at se at brugeren får tildelt det navn som bliver indtastet i opstartsapplikationen, hvilket kan ses, ved at der sættes breakpoint ved linje 32 i GameViewModel.cs-filen. Resultatet hertil adresseres i afsnittet Resultater. Desuden har gruppen ikke anvendt integrationstest, da dette ikke giver logisk mening uden udarbejdelse af Unit test. Derudover har gruppen ikke været beredt med de rette værktøjer herunder en C.I.-server, da fagligheden manglede og dette emne blev undervist i forholdvist sent i semestret.

9.3 Roulette - Test

I dette afsnit vil der være en gennemgang på testen i Roulette-applikationen. En kort beskrivelse og forklaring på de relevante test vil være gennemgået. Endvidere vil der være en forklaring på hvorfor unit- og integrationstest ikke er inddraget i testen.

Det er valgt at vise en test på User Story 4 og User Story 1. På baggrund af de ensartet scenarier, testes kun en enkel sekvens på koden, som tager udgangspunkt i sekvensdiagrammet, som findes i afsnittet Softwaredesign. Med samme princip hos Blackjack-applikationen, skal testen kunne vise at spilleren har et navn og får tildelt deepcoins

fra opstartsapplikationen. Via debuggeren vises de enkelte dele af applikationen for at tjekke om deepcoins, der tilhænger spilleren, bliver anvendt igennem Roulette-applikationen.

Det visuelle test af applikationen er foretaget igennem accepttesten. Men det interne funktionalitet vil være vist under afsnittet Resultater. Det vil være testet via debuggeren, for at vise om de forskellige attributter indeholder de rigtige parametre afhængig af hvad der foretages på applikationen.

Unit- og integrationstest er ikke blevet foretaget på baggrund af den simplificerede version af Roulette-applikation. Begrundelsen til dette, ligger under afsnittet Diskussion.

9.4 Opstartsapplikation - Test

Med udgangspunkt i implementering af opstartsapplikationen og User story 5 "Brugeren skal kunne vælge mellem at spille BlackJack eller Roulette", kommer gruppen i dette afsnit ind på hvilke af applikationens planlagte funktionalitet virker. Gruppen gennemgår de moduler, der kunne testes og hvorledes testene blev foretaget.

Det første der ønskes testet for opstartsapplikationen er om brugeren fra SelectGameWindow, kan vælge at spille BlackJack, hvorefter at BlackJack-vinduet skal åbnes ved tryk på en knap. Det andet der skal testes, er om Roulette-vinduet åbnes ved tryk på den rette knap. Begge dele testes visuelt ved kørsel af applikationen. Er det tilfældet at vinduet skiftes, består funktionaliteten testen.

10 Resultater

Her formidles projektets resultater fra accepttesten kort. Ligesom i de andre afsnit opdelt i emnerne database, BlackJack, Roulette og opstartsapplikation. I afsnittet kommer gruppen også ind på projektets resultater fra accepttesten, hvilket kan ses på tabellen neden for.

Accepttest	Vurdering
1. Brugeren skal kunne indtjene deepcoins ved at vinde spil.	Godkendt
2. Brugeren skal kunne få opbevaret deepcoins på sin konto i databasen.	Delvist godkendt
3. Brugeren skal kunne spille blackjack	Godkendt
4. Brugeren skal kunne spille roulette	Godkendt
5. Brugeren bør kunne vælge mellem at spille BlacJack eller Roulette.	Godkendt
6. Brugeren bør kunne oprette en brugerkonto selv.	Godkendt
7. Brugeren bør kunne opleve visuel animation i spillene.	Godkendt
7. Brugeren kan have en adgang til en virtuel butik (The DeepShop).	Ikke godkendt
8. Brugeren kan kunne bruge sine deepcoins på præmier i The DeepShop.	Ikke godkendt
9. Brugeren kan kunne læse reglerne for spillene i applikationen.	Ikke godkendt
10. Brugeren kan få gemt sin highscore for et vist spil.	Ikke godkendt
12. Brugeren kan opleve at spillene har lydeffekter.	Ikke godkendt
13. Brugeren kan få applikationen fremvist på en HTML-hjemmeside,	Ikke godkendt

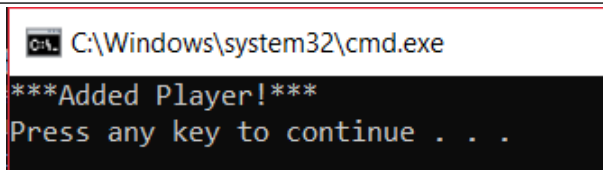
10.1 Database

Under dette afsnit vil der blive givet nogle eksempler på resultater, der fremkommer under tests af de operationer, som indgår i databasen. Operationerne, der vil blive gennemgået, er følgende: Create, Read, Update og Delete.

10.1.1 Create

Create-operationens testresultater uddybes under dette punkt. Der er blevet implementeret noget kode, som har til formål at oprette en ny player og gemme det i databasen. For at kunne oprette en ny player, skal DCAApp.cs anvendes. I denne source-fil findes en funktion, der tager parametre, som er relevante for oprettelse af en ny player. Som parametre angives et brugernavn (UserName), en score (HighScore) og en pengepung (Wallet). Når disse parametre er korrekt angivet, angives det i DCAApp.cs, at det er AddPlayerDB-metoden, der skal køres.

Når programmet eksekveres, vil et terminalvindue dukke op, som meddeler, at en ny player er blevet tilføjet til databasen.



```
C:\Windows\system32\cmd.exe
***Added Player!***
Press any key to continue . . .
```

Figur 25: Terminalvindue udskrift

Når meddelelsen modtages via terminalvinduet, bliver der tjekket efter i tabellerne, som hører til databasen, her kan det bekræftes, at en ny player er blevet oprettet. Billedet nedenunder illustrerer dette.

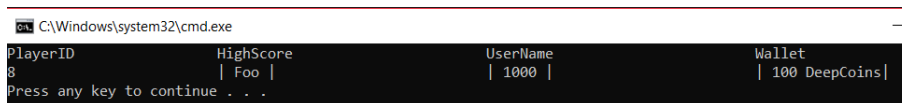
	PlayerID	HighScore	UserName	Wallet
	8	1000	Foo	100

Figur 26: Tabeloversigt for Databasen

Da det nu kan bekræftes, at den nye player er oprettet og befinder sig i databasen med dens tilhørende værdier, skal næste operation testes, hvilket er Read-operationen, der har til formål at hente en player fra databasen med dens tilhørende værdier og udskrive det til terminalvinduet.

10.1.2 Read

Resultaterne for Read-operationen kan læses under dette punkt. Read-operationen fungerer som en get-metode. Der er blevet implementeret noget kode, som kan hente en player med dens værdier fra databasen. Read-operationen testes ved at kalde på en funktion, som er i DeepCasinoDBUtil.cs, hvor andre funktioner også er. Read-operationen vil fungere uanset, om der findes data gemt i databasen, eller om der ikke er data gemt i databasen, operationen udskriver blot en tom tabel, hvis databasen ikke indeholder data. For at teste at metoden faktisk virker, arbejdes der videre med dataene fra forrige punkt. Programmet eksekveres og følgende terminalvindue vil dukke op:



```
C:\Windows\system32\cmd.exe
PlayerID      HighScore      UserName      Wallet
8             | Foo |         | 1000 |         | 100 DeepCoins|
Press any key to continue . . .
```

Figur 27: Terminaludskrift af Read-operationen

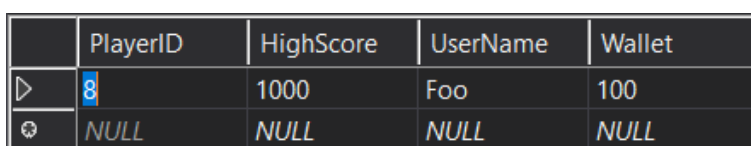
Read-operationen virker som forventet, det kan altså konkluderes, at det data, der findes i databasen, kan blive udskrevet ved brug af Read-operationen.

Oplysninger, som ønskes i databasen, kan tilføjes via Create-operationen og for at bekræfte, at oplysningerne sættes ind i databasen, skal de kunne hentes med en get-metode, som gruppen har kaldt Read-operation. De indtastede oplysninger skal også kunne opdateres. Opdatering af oplysninger kan bl.a. findes sted, når en player spiller videre på sin bruger og optjener eller taber points. Til opdatering af oplysninger er det blevet implementeret en Update-operation, som har til formål at opdatere dataene til en player.

10.1.3 Update

Under dette punkt bliver Update-operationens resultater uddybet. Update-operationen har til formål at opdatere data i databasen, som tilhører en player. Det er en forudsætning, at der skal være en player oprettet, før Update-operationen kan anvendes. Update-operationen testes ved at kalde på en funktion, som findes i DeepCasinoDBUtil.cs, som kører en opdatering på en bestemt player. Det er en nødvendighed, at der skal være kendskab til den ønskede player's ID, før Update-operationen kan anvendes. Når en player skal opdateres, skal ID'et angives i player-objektet, som kan findes i DCAApp.cs, her skal de resterende parametre også angives til det, som de skal ændres til.

På billedet forneden illustreres et billede, der viser, hvad databasen på nuværende tidspunkt indeholder. Her kan det ses, at fra punktet hvor Create-operationen blev testet, blev der indsat nogle værdier i databasen, disse værdier findes stadig i databasen.



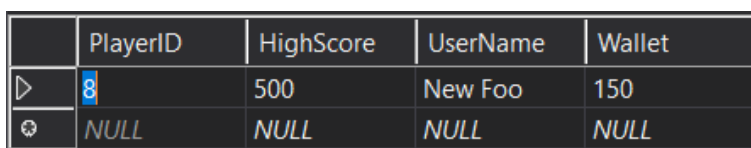
	PlayerID	HighScore	UserName	Wallet
▶	8	1000	Foo	100
⚙	NULL	NULL	NULL	NULL

Figur 28: Billede af tabel før Update-operationen køres

Da det kan på baggrund af ovenstående billede konkluderes, at de indtastede oplysninger fra punktet med Create stadig findes i databasen, kan Update-operationen anvendes til at ændre på dataene.

For at anvende Update-operationen skal ID'et på den ønskede player sættes ind som parameter i player-objektet, som befinder sig i DCAApp.cs, derefter skal de andre parametre ændres til det ønskede.

Programmet eksekveres og der tjekkes efter i databasen tabel, om oplysningerne faktisk er ændret. Billedet forneden illustrere et billede af tabellen, efter Update-operationen er kørt:



	PlayerID	HighScore	UserName	Wallet
▶	8	500	New Foo	150
⚙	NULL	NULL	NULL	NULL

Figur 29: Billede af tabel efter Update-operationen køres

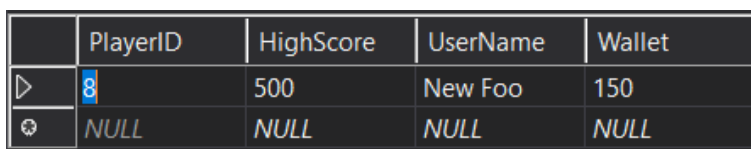
På baggrund af ovenstående billede kan det konkluderes, at oplysningerne, som findes i databasen, ændres til de nye informationer, som angives i player-objektet i DCAApp.cs. Det kan også konkluderes, at det er den samme player, da ID'et stemmer overens på de to billeder, som illustreres foroven.

Da det nu kan konkluderes, at Update-operationen virker som forventet, skal den sidste operation testes, hvilket er Delete-operationen. Delete-operationen skal fjerne en player fra databasen med dens tilhørende værdier.

10.1.4 Delete

Under dette punkt kan Delete-operationens resultater ses. Delete-operationen har det formål, at det skal slette en player fra databasen. Det er en forudsætning, at en player skal være oprettet i databasen, før Delete-operationen kan anvendes. Der skal være kendskab til det ID, som en player har, før den ønskede player kan slettes fra databasen. For at anvende Delete-operationen skal ID'et på den ønskede player sættes ind som parameter i player-objektet, som findes i DCAApp.cs, blot ID'et er tiltrækkeligt for at slette en player.

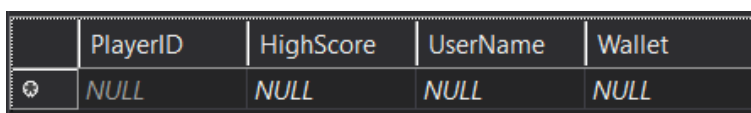
Der skal tjekkes efter, om der findes oplysninger i databasen, dette gøres ved at kigge på tabellerne i databasen. Billedet forneden illustrerer de oplysninger, som findes i databasen.



	PlayerID	HighScore	UserName	Wallet
▶	8	500	New Foo	150
⚙	NULL	NULL	NULL	NULL

Figur 30: Billede af tabel før Delete-operationen køres

På baggrund af ovenstående billede kan det konkluderes, at de opdaterede værdier, som blev indsat under forrige punkt med Update-operationen, stadig findes i databasen. Da det nu er konkluderet, kan Delete-operationen blive kørt. Billedet forneden illustrerer databasen efter, Delete-operationen er blevet eksekveret.



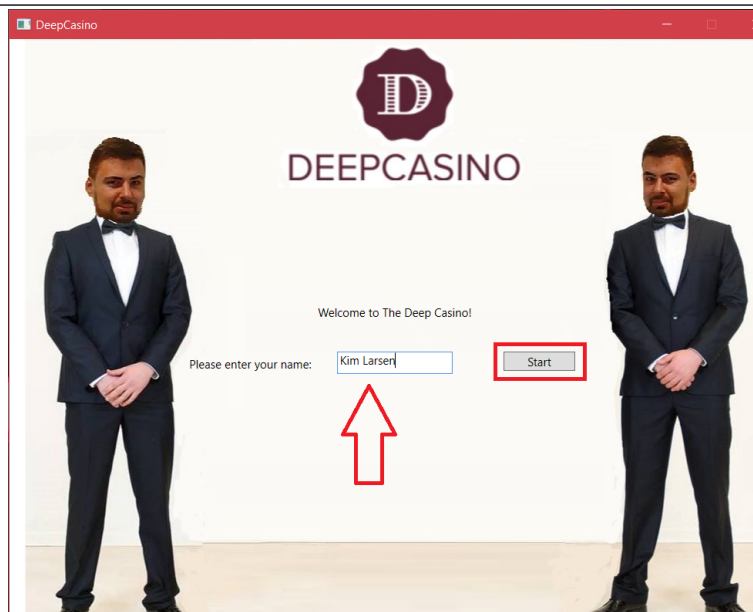
	PlayerID	HighScore	UserName	Wallet
⚙	NULL	NULL	NULL	NULL

Figur 31: Billede af tabel efter Delete-operationen køres

Det kan på baggrund af ovenstående billede konkluderes, at Delete-operationen virker, som det er forventet. De indtastede oplysninger fjernes helt fra databasen, når Delete-operationen er kørt.

10.1.5 Forbindelsesledet mellem databasen og DeepCasino - Test

Resultatet af forbindelsesledet mellem databasen og Deepcasino kan læses under dette punkt. Forbindelse mellem databasen og DeepCasino fungerer vha. SQL-forbindelsen, hvor der er implementeret SQLCommand til at kunne udskrive på tabellen i databasen. Billedet nedenunder ses der et billede af startsiden af applikationen, hvor man indtaster et navn på Textbox, som er markeret med rødt-pil. Derefter trykker man på start-knappen, hvor der så vil ske en event.



Figur 32: Billedet af start-siden af applikationen

Billedet nedenunder ses der en tabel fra databasen, hvor man kan se brugerens UserName og DeepCoins.

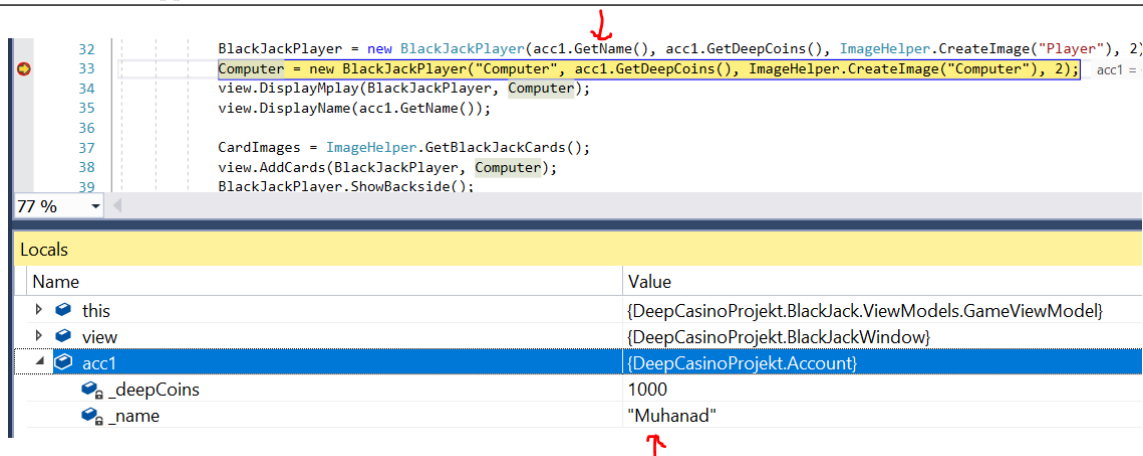
	PlayerID	HighScore	UserName	Wallet
▶	22	0	Kim Larsen	1000
⚙	NULL	NULL	NULL	NULL

Figur 33: Billede af tabellen fra database

10.2 BlackJack

Det vil i denne del være oplagt at give eksempler på resultater af Unit Tests af de forskellige metoder, men da disse ikke er udarbejdet, har gruppen foretaget sig test vha. debug, hvilket er yderligere beskrevet i afsnittet Test.

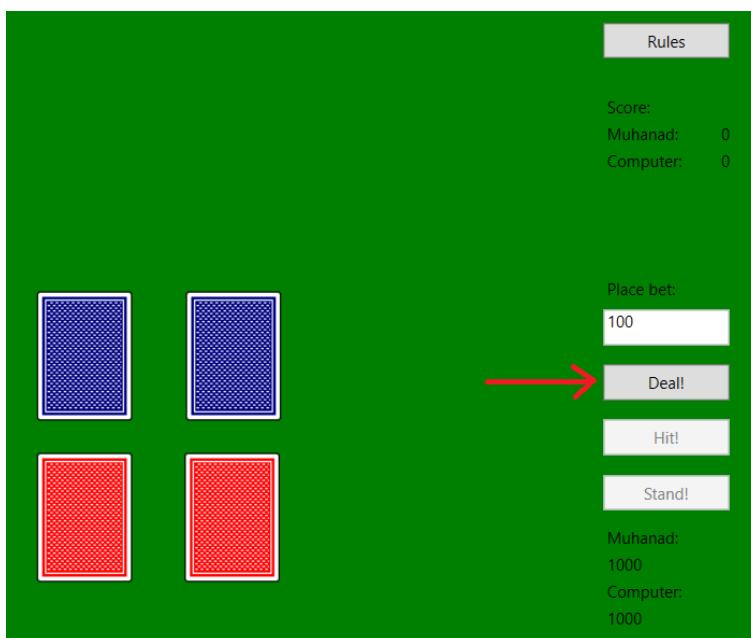
På figur 34 illustreres et eksempel på, hvordan gruppen har brugt debugging for at verificere at navn samt deepcoins bliver tildelt brugeren. De relevante steder at kigge på figuren peges på med to røde pile. Den øverste pil viser den kodelinje, hvori gruppen har sat et breakpoint. Den nederste pil viser det sted man ser, at brugerens navn "Muhanad" og deepcoins er kommet med over i BlackJack-projektet fra opstartsapplikationen. Brugeren Muhanads deepcoins vil også dekrementeres eller inkrementeres alt efter om Muhanad vinder eller taber et spil.



Figur 34: Eksempel på brug af debug.

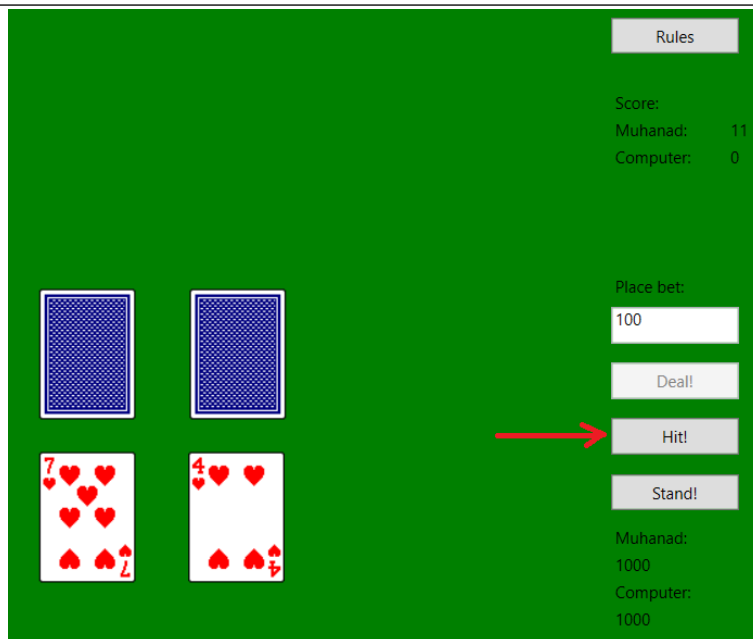
Følgende ønsker gruppen at demonstrere at User Story 3 "Brugeren skal kunne spille BlackJack", fungerer efter hensigten.

På figur 35 ses BlackJack-applikationen efter at projektet køres. Den røde pil der peger på "Deal-knappen, er der for at illustrere at den trykkes på for at opnå funktionalitet på figur 36.



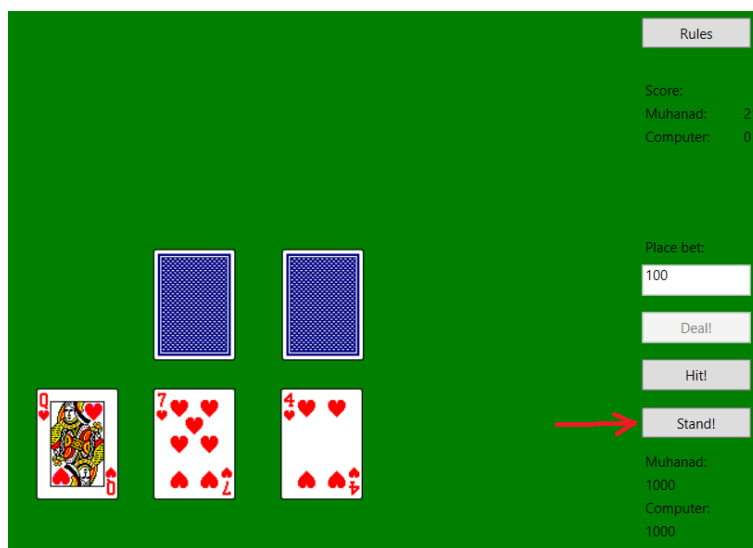
Figur 35: Ved kørsel af BlackJack-applikationen.

Som det ses på figur 36 får brugeren Muhanad tildelt to kort. Disse kort har randomiserede værdier for hver kørsel med passende billeder. Disse værdier bliver lagt sammen i en counter, som ses på højre side af figuren. Den røde pil på figuren, viser at når knappen "Hit!" trykkes på vil man opnå funktionaliteten som illustreret på figur 37.



Figur 36: BlackJack-applikationen efter at brugeren trykker "Deal".

På figur 37 ses at brugeren har fået tildelt et nyt kort og at counteren (på højre side af figuren) er opdateret. Pilen på figuren peger på "Stand!"-knappen som er en indikation på, at hvis den trykkes på, ses funktionalitet vist på 38.



Figur 37: BlackJack-applikationen efter at brugeren trykker "Hit"

Her (figur 38) vises, at dealeren (CPU) selv får tildelt kort. Opnår dealeren ikke mere end brugeren, annonceres at brugeren har vundet, som vist på figuren ved den røde pil.



Figur 38: Ved kørsel af BlackJack-applikationen.

Herfra kan gruppen godkende at testen af User Story 3 er gennemgået og fungerer efter hensigten mht. de essentielle funktioner, der dækker gruppens formål med BlackJack.

Mht. statistisk analyse ses det på figur 39 at gruppen har formået at holde maintainability indekset markant over 20.

Hierarchy	Maintainability Index	Cyclomatic Compl...	Depth of Inheritance	Class Coupling	Lines of Code
DeepCasinoProjekt (Debug)	86	190	9	62	379
BlackJack.Commands	86	22	1	7	34
DeepCasinoProjekt	86	36	9	27	74
DeepCasinoProjekt.BlackJack.Helpers	72	23	1	10	44
DeepCasinoProjekt.BlackJack.Models	91	4	1	5	7
DeepCasinoProjekt.BlackJack.ViewModels	69	43	1	26	120
DeepCasinoProjekt.Interfaces	100	7	0	1	0
DeepCasinoProjekt.Models	91	17	1	3	25
DeepCasinoProjekt.Roulette	65	25	9	16	68
DeepCasinoProjekt.Roulette.Interfaces	100	8	0	0	0
DeepCasinoProjekt.Roulette.Models	93	5	1	1	7

Figur 39: Visual studio statistisk analyse

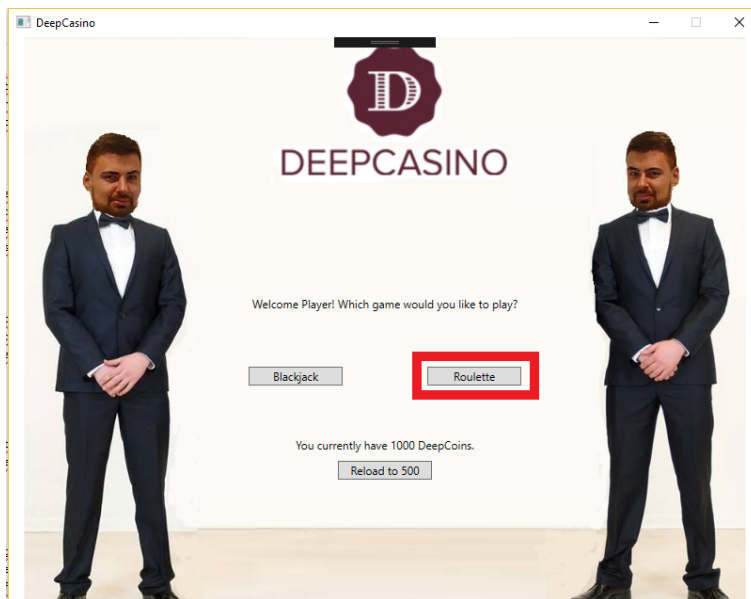
Med udgangspunkt i accepttesten og resultaterne derfra, har gruppen demonstreret funktionalitet dækkende for User story 1 og 3, men også User story 7 der lyder "Brugeren bør kunne opleve visuel animation i spillene". BlackJack-applikationens skift af kort fyldestgøre denne visuelle oplevelse for brugeren.

10.3 Roulette

I dette afsnit vil der være gennemgået resultaterne i forhold til hvordan test-delen er blevet udført. Her fremgår en række figurer, der viser den interne funktionalitet, inklusivt korte beskrivelser. Billederne vil generelt bestå af debug og breakpoints på koden, samt det visuelle. Beskrivelserne er korte, da de allerede er beskrevet uddybende under

afsnittet Softwaredesign og Software implementering

Figur 40 nedenfor, viser opstartsapplikationen. Meget kort illustrerer figuren, hvordan applikationen tilhænger spilleren's konto til Roulette-applikationen. De senere figurer viser gennemgangen af de få steps via debuggeren.



Figur 40: Spiller der vælger Roulette på opstartsapplikationen

Figur 41 og figur 42 nedenfor, er en forlængelse af den øvrige figur. Meget simpelt, viser de **case 2**, som er anvendelsen af RouletteWindow.

```

1 reference
50 public RouletteWindow(Account acc) acc = {Account}
51 {
52     _account = acc;
53     InitializeComponent();
54
55     RoulettePlayer = new RoulettePlayer(acc.GetName(), acc.GetDeepCoins()); Account.GetName() = "Player", Account.GetDeepCoins() = 1000
56     _wallet += _account.GetDeepCoins(); ≤ 2ms elapsed _wallet = 0, _account = {Account}
57     UpdateWallet();
58 }

```

Figur 41: Spiller bliver registreret i Roulette-applikationen

```

18 public void SetContent(Account p, int contentId) p = {Account}, contentId = 2
19 {
20     switch (contentId)
21     {
22         case 0: ContentHolder.Content = new SelectGameWindow(p); break;
23         case 1: ContentHolder.Content = new BlackjackWindow(p); break;
24         case 2: ContentHolder.Content = new RouletteWindow(p); break; ≤ 1ms elapsed ContentHolder.Content = {SelectGameWindow}, p = {Account}
25     }
26 }

```

Figur 42: Case 2. Applikationen arbejder med RouletteWindow

Kort efterfulgt fra opstartsapplikationen, ses forsiden til Roulette-applikationen. Spilleren får mulighed at vælge en værdi spilleren ønsker at satse, samt hvad spilleren ønsker at spille på. Det ses på figur 43 nedenfor, at spilleren vælger **bet 10** og **black**.



Figur 43: Forsiden på Roulette-applikationen - RouletteWindow

Det næste skridt er gennemgangen af spillet roulette. Figur 44 nedenfor viser resultatet, efter spilleren har trykket på Spin knappen. En beskrivende sekvens på dette, kan læses under afsnittet Softwaredesign.



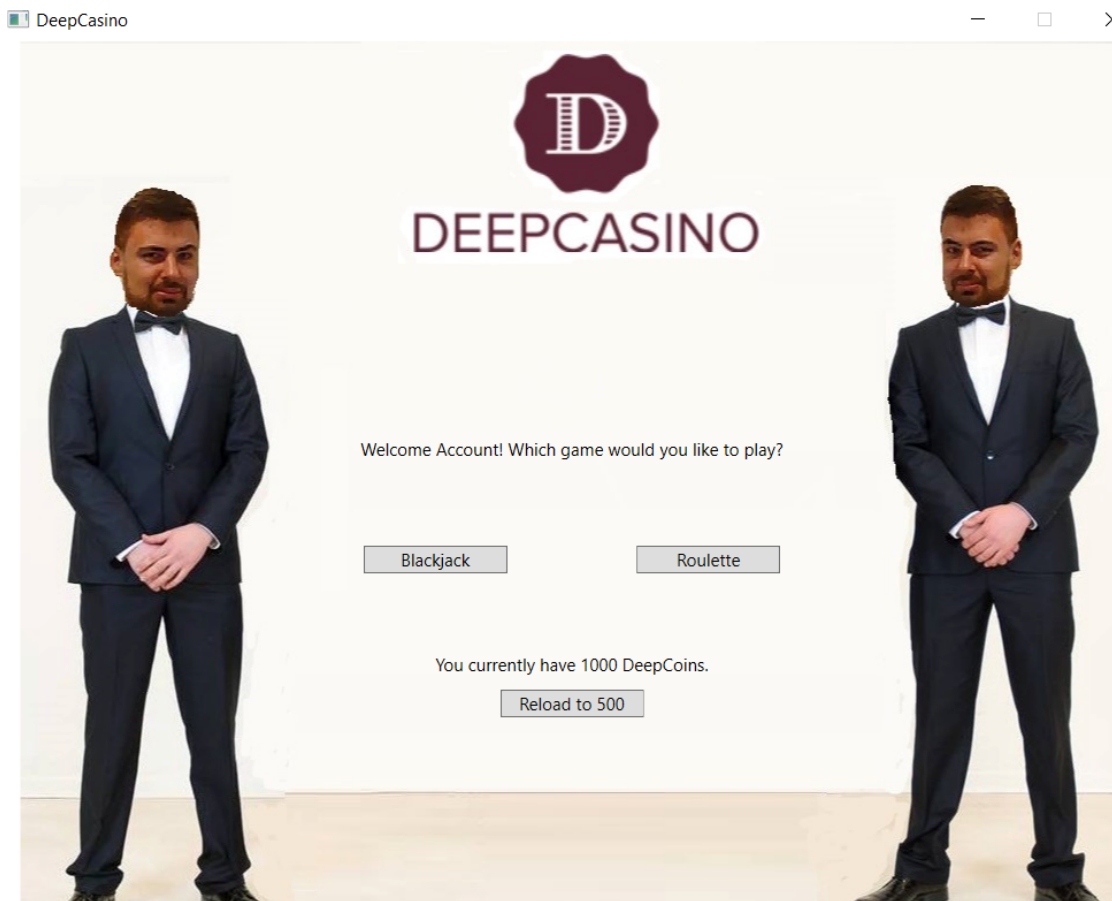
Figur 44: Resultatet på roulette, efter applikationen er kørt

Som beskrevet under afsnittet Softwaredesign for Roulette, så eksekveres de resterende principper i applikationen på

samme måde.

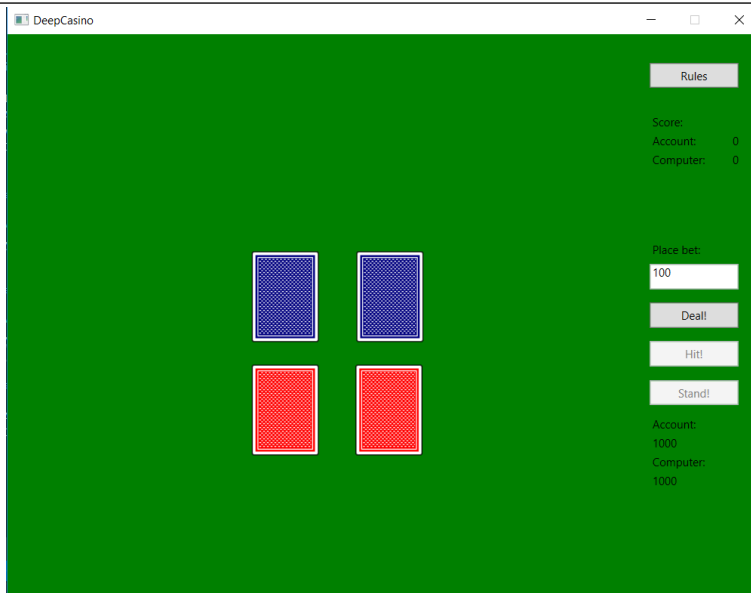
10.4 Opstartsapplikation

Her vises resultaterne til opstartsapplikationen fra de test som blev beskrevet i afsnittet Test. Figur 45 viser applikationens SelectGameWindow, hvori man kan se at knapperne til BlackJack og Roulette er indeholdt.



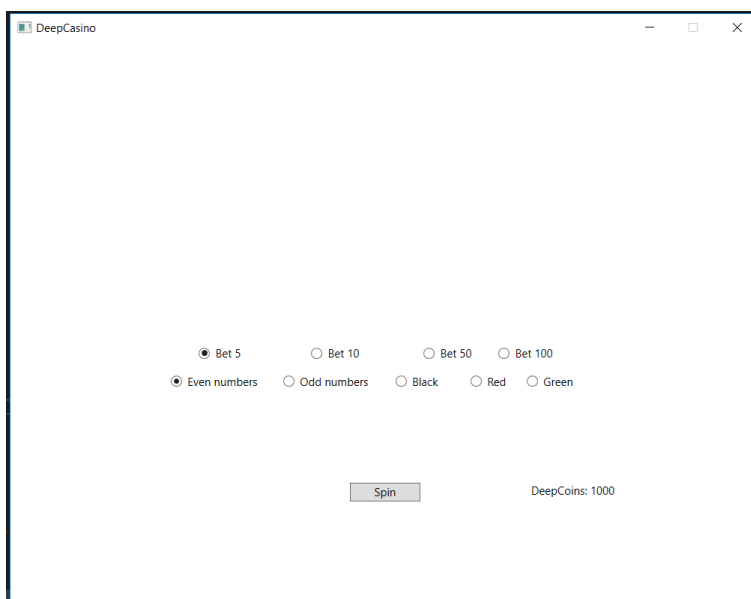
Figur 45: SelectGameWindow

Når gruppen fra SelectGameWindow(figur 45) trykker på Blackjack-knappen åbnes vinduet som ses på figur 46, hvilket var det forventede resultat.



Figur 46: BlackjackWindow

Ligeledes når gruppen fra SelectGameWindow(figur 45) trykker på Roulette-knappen åbnes vinduet som ses på figur 47, hvilket var det forventede resultat.



Figur 47: RouletteWindow

I forhold til acceptttesten er opstartsapplikationen fyldestgørende for User story 5. Ud fra de foretagne test vurderes User story 5 som godkendt.

11 Diskussion af resultater

Under dette afsnit udpeges og diskuteres relevante dele af de opnåede resultater og deres betydning i de forskellige dele af projektet. Gruppen giver en samlet vurdering af de opnåede resultater med relation til projektets problemformulering.

11.1 Database

Under udviklingen af databasen har det været svært at få det integreret med The Deep Casino, eftersom man ikke kan integrere to solutions med hinanden. Som man kan se ud fra modul testene for databasen, virker databasen som den skal. Dog har der været nogle problemer undervejs mht. nogle compiler fejl, som heldigvis er blevet fikset ved at slette obj mappen for database projektet og derefter rebuildet solution.

Der har i starten været diskuteret om, at databasen skulle være på en lokal database, hvilket resulterede i, at man ikke kunne integrere de to solutions sammen. Derfor blev det vedlagt senere hen at lave det på en server som underviseren i I4DAB heldigvis havde til rådighed. Connection string til databasen blev så ændret til en database som er på serveren, hvilket betyder, at man kan tilgå databasen fra forskellige pc'er, så længe man er logget ind på den rigtige database. Problemet med CRUD operationerne for The Deep Casino projekt er nu, at den ikke fungerer helt som den skal, da Create operationen er den eneste som er implementeret i The Deep Casino projekt, eftersom gruppen ikke har haft tid til at implementere de andre operationer. Når en bruger indtaster et brugernavn der allerede findes i databasen, vil databasen oprette en ny række med samme navn, hvilket ikke er hensigten. Dette problem har der heller ikke været tid til at fikse grundet tidsmangel.

11.2 BlackJack

Under udviklingen af BlackJack-applikationen har gruppen haft forskellige udfordringer. Det lykkedes gruppen at løse mange af disse udfordringer, men ligeledes er der få som bliver adresseret under fremtidigt arbejde. Hovedformålet for BlackJack-applikationen er at User story 3 bliver realiseret. Om gruppen har opnået at realisere denne user story, er vist i accepttesten, og er vurderet som godkendt, baseret på gruppens test og resultater.

Gruppen er opmærksomme på at løsningen kunne være lavet på et utal af måder. Eksempelvis har gruppen overvejet om hvorvidt der skulle sættes tid af til, at implementere RelayCommand i vores applikationer til anvendelse af ICommand. Det vil have været en markant fordel at implementere, hvis gruppens applikation bestod af flere kommandoer, men da dette ikke var tilfældet valgte gruppen at sætte fokus og tid af til de allerede eksisterende kommandoers funktionalitet for at opfylde User story 3.

Gruppen har ligeledes diskuteret om den grafiske brugergrænseflade var tilstrækkelig nok, og om den illustrerer projektets formål. Gruppen har her specielt haft fokus på User story 7 "Brugeren bør kunne opleve visuel animation i spillene", hvilket gruppen vil sige er implementeret og fyldestgørende nok til godkendelse på baggrund af opnåede resultater.

Der er dog få user stories, som gruppen ikke har fået fuldført herunder at kunne læse reglerne for det pågældende

spil, user story 9. Dette ser gruppen dog ikke som en vanskelig opgave at få implementeret, men er blevet tilsidesat da denne del ikke har ligeså høj prioritet som de nævnte user stories. Derudover er det heller ikke lykkedes gruppen at opleve lydeffekter i applikationen af samme årsag.

11.3 Roulette

Det var planlagt af gruppen at anvende et design pattern, som henholdsvis var MVVM. Dette skulle implementeres på begge spil-applikationer, men på baggrund af den manglende tid og ikke-funktionelle kode, måtte det desværre annulleres. Gruppen stræbte istedet efter at få udviklet en velfungerende applikation, hvor den stadig overholdte de angivet User Stories og FURPS principper.

Endvidere var det tiltænkt, at der skulle udføres en række Unit Tests i forbindelse med applikationen, men efter en vurdering af denne, fandt gruppen det tidspresset og mere kompliceret, da testene ikke ville være specielt store, eftersom at MVVM pattern ikke var implementeret.

På baggrund af det sene valg om at simplificere koden, har det ikke været muligt for gruppen at køre unit test og integrationstest over applikationen. Blandt andet skyldes det også de manglende lag af design for kodestrukturen, som havde til formål at danne en lav-kobling for koden.

11.4 Opstartsapplikation

Med udgangspunkt i resultaterne fra accepttesten, er det User story 5, som inddrager opstartsapplikationen. Baseret på testene der er blevet foretaget i forbindelse med opstartsapplikationen og de efterviste resultater, godkendes User story 5. Der findes andre måder at teste gruppens applikation på, eksempelvis ved at bruge debug hvor man kan følge alle variabler, og se at den specifikke variabel bliver ændret, når den skal. Gruppen har foretaget debug med breakpoints og set at de vigtigste variable ændres – dog er dette ikke dokumenteret grundet tidsnød. Gruppens løsning på denne del har ført til en række fordele bl.a. at designet af løsningen har gjort det nemmere, at implementere flere Windows eller UserControls. Gruppen overvejede om switch-case-delen i MainWindow skulle være foretaget i XAML kode, dvs. at der ikke bruges en switch-case overhovedet til at skifte mellem vinduer. Ulempen er dog den høje kobling. Gruppen tog i betragtning at DeepCasino ikke indeholder flere end to spil, så vil det være mere overskueligt at skiftet mellem applikationerne via switch-casen, der er angivet i MainWindow.cs.

12 Ordliste

Ord	Betydning
BlackJack	At hit-, stand- og deal-funktionen implementeret.
Business logic	
C.I-server	
Code coverage	
DeepCoins	Virtuel valuta tilknyttet The Deep Casino.
DeepShop	Virtuel Butik tilknyttet The Deep Casino.
SRP	Single Responsibility Principle.
OCP	Open-Closed Principle.
LSP	Lisskov's Substitution Principle.
ISP	Interface Segregation Principle.
DIP	Dependency Inversion Principle.
GUI	Kurset GUI (Graphical User Interface) programmering.
Løs kobling	
Presentation logic	
ReSharper	Et plugin til Visual Studio.
Roulette	
Startspil	Forløb hvor en BlackJackPlayer starter et spil og får tildelt sine første kort.
Visuel animation	Brugergrænseflade der inderholder knapper, resultater af spil, evt. kort og chips.

13 Referenceliste

13.1 Database referenceliste:

<https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqlcommand?view=netframework-4.7.2>

13.2 BlackJack referenceliste:

Aarhus University School Of Engineering - Electro and Information Communication Technology Department: The Model-View-ViewModel.

<https://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/>

<https://blog.codecentric.de/en/2011/10/why-good-metrics-values-do-not-equal-good-quality/>

<https://blogs.msdn.microsoft.com/zainnab/2011/05/26/code-metrics-maintainability-index/>

<https://www.tutorialspoint.com/mvvm/>

<https://msdn.microsoft.com/en-us/magazine/dn237302.aspx>

13.3 Roulette referenceliste:

13.4 Opstartsapplikation referenceliste:

GUI lektion 02 - C and WPF Intro

GUI lektion 03 - XAML and Layout

GUI lektion 04 - Input and Event routing

GUI lektion 05 - Controls and Application

GUI lektion 06 - Data binding and Commands