

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: _____

Vejleder: Lars Mortensen

Gruppens GitHub link:

Antal anslag: 71.681

<https://github.com/DannyTraan/I4PRJ4>

Resumé

Denne rapport fokuserer på at beskrive gruppe to selvvalgte projekter på fjerde semester på IKT studiet ved Aarhus Universitets Ingeniørhøjskole. Det valgte projekts formål er at designe samt implementere et spil i wpf format med en dertilhørende database. Dette spil blev udformet som en idé i form af et offline casino platform, der indeholder to spil: black jack og roulette, som blev valgt i projektets startfase.

Gruppen valgte at arbejde iterativt, hvilket gjorde udviklingsprocessen væsentlig lettere i forhold til test og ændringer i løbet af arbejdesprocessen. Denne tilgang til projektets arbejde var blevet anvendt i tidligere semester projekter med gode erfaringer. I forhold til arbejdsflow blev der anvendt frameworket SCRUM på grund af dets enkle tilgang. Dog havde dette ikke helt det ønskede mål, som forventet men på trods af dette lykkedes det gruppen at opnå hovedformålet med projektet, hvilket var at etablere gruppens eget design og implementering af casino platformen og dets tilhørende spil.

Alt i alt taget i betragtning var spillets design og implementation en success, omend med nogle få problemer. Disse havde dog ikke nogle effekt på den overordnede performance af applikationen, når denne blev kørt.

Abstract

This paper aims to provide an in-depth description of a self chosen project for the fourth semester in the IKT major at Aarhus University's School of engineering. The chosen project by the students in the group seeks to design and implement a game in wpf format with an accompanying database. With this in mind, an offline casino platform consisting of two games: blackjack and roulette was agreed upon in the preliminary phase.

To handle the work process, the group chose to approach the project workload in a iterative way, as the group had experience with this approach from previous semester projects, thus making the development process easier in terms of testing and making changes throughout the process. As for the actual framework for the workflow, the group used SCRUM granted this did not have the most favorable outcome as intended. Having said that the group was able to follow through with main purpose of the project, which was to establish the group's own design and implementation for casino platform and its corresponding games .

All things considered the game design and implementation was successful although with some minor problems, however these did not affect the overall performance when the application was run.

Indholdsfortegnelse

| | |
|---|-----------|
| 1 Forord | 6 |
| 2 Indledning | 6 |
| 2.1 Problemformulering | 6 |
| 2.1.1 Projektbeskrivelse | 6 |
| 2.2 Projektafgrænsning | 7 |
| 2.2.0.1 Projektets øverste mål | 7 |
| 3 Kravspecifikation | 8 |
| 3.1 Funktionelle krav | 8 |
| 3.2 Ikke-funktionelle krav | 8 |
| 3.3 Accepttest beskrivelse med brug af Gherkin: | 9 |
| 4 Arbejdsfordeling | 11 |
| 5 Metode og proces | 12 |
| 6 Analyse | 13 |
| 6.1 Database | 13 |
| 6.2 Blackjack | 13 |
| 6.3 Roulette | 14 |
| 6.4 Opstartsapplikation | 14 |
| 7 Softwarearkitektur | 15 |
| 7.1 Domænemodel | 15 |
| 7.2 Applikationsmodel | 16 |
| 7.2.1 User story 3 - Blackjack | 16 |
| 7.2.1.1 Klassediagram | 17 |
| 7.2.1.2 Sekvensdiagram | 18 |
| 7.2.2 User story 4 - Roulette | 18 |
| 7.2.2.1 Klassediagram | 18 |
| 7.2.2.2 Sekvensdiagram | 19 |
| 7.2.3 User story 5 - Opstartsapplikation | 20 |
| 7.2.3.1 Klassediagram | 20 |
| 7.2.3.2 Sekvensdiagram | 20 |

| | | |
|-----------|---|-----------|
| 8 | Softwaredesign | 21 |
| 8.1 | BlackJack | 21 |
| 8.1.1 | Klassediagram | 21 |
| 8.1.2 | Sekvensdiagram | 22 |
| 8.2 | Roulette | 23 |
| 8.2.1 | Klassediagram | 23 |
| 8.2.2 | Sekvensdiagram | 24 |
| 8.3 | Opstartsapplikation | 25 |
| 8.3.1 | Klassediagram | 25 |
| 8.3.2 | Sekvensdiagram | 26 |
| 9 | Software implementering | 27 |
| 9.1 | Database | 27 |
| 9.1.1 | CRUD operationer | 28 |
| 9.1.2 | Forbindelseledet mellem applikationen og Database | 30 |
| 9.2 | BlackJack | 31 |
| 9.2.1 | Statisk analyse og Software matrice | 33 |
| 9.3 | Roulette | 34 |
| 9.3.1 | Private properties | 35 |
| 9.3.2 | spinButton_Click() | 35 |
| 9.3.3 | Bet() | 36 |
| 9.3.4 | WinOrLose() | 36 |
| 9.4 | Opstartsapplikation | 38 |
| 10 | Test | 40 |
| 10.1 | Database - Test | 40 |
| 10.1.1 | CRUD operationer - Test | 40 |
| 10.1.1.1 | Create - Test | 40 |
| 10.1.1.2 | Read - Test | 40 |
| 10.1.1.3 | Update - Test | 41 |
| 10.1.1.4 | Delete - Test | 41 |
| 10.1.2 | Forbindelsesledet mellem databasen og DeepCasino - Test | 41 |
| 10.2 | BlackJack - Test | 41 |
| 10.3 | Roulette - Test | 42 |
| 10.4 | Opstartsapplikation - Test | 43 |
| 11 | Resultater | 44 |
| 11.1 | Database | 44 |
| 11.1.1 | Create | 44 |

| | |
|--|-----------|
| 11.1.2 Read | 45 |
| 11.1.3 Update | 45 |
| 11.1.4 Delete | 46 |
| 11.1.5 Forbindelsesledet mellem databasen og DeepCasino - Test | 47 |
| 11.2 BlackJack | 48 |
| 11.3 Roulette | 51 |
| 11.4 Opstartsapplikation | 53 |
| 12 Diskussion af resultater | 56 |
| 13 Konklusion | 57 |
| 14 Fremtidigt arbejde | 58 |
| 14.1 Database | 58 |
| 14.2 BlackJack | 58 |
| 14.2.1 MVVM | 58 |
| 14.2.2 Tests | 59 |
| 14.3 Roulette | 59 |
| 14.4 Opstartsapplikation | 59 |
| 15 Ordliste | 60 |
| 16 Referenceliste | 61 |
| 16.1 Database referenceliste: | 61 |
| 16.2 BlackJack referenceliste: | 61 |
| 16.3 Roulette referenceliste: | 61 |
| 16.4 Opstartsapplikation referenceliste: | 61 |

1 Forord

I denne rapport gennemgås semesterprojektet på 4. semester for Informations- og kommunikationsteknologi på Ingeniørhøjskolen Aarhus Universitet. Gruppen har haft til opgave at lave et produkt og en dertilhørende rapport, hvori der er lagt særlig fokus på anvendelse og inddragelse af teorier fra semestrets kurser og brug af processtyring. Rapportens skribenter er de otte personer, der ses på forsiden, hvor alle har det tilfældes, at de studere på Aarhus Universitet og tager uddannelsen Informations- og Kommunikationsteknologi.

Derudover består dette projekt af en dokumentationsrapport, dokumentationsbilag i forbindelse med udviklingen af produktet og en prototype i form af en Visual Studio solution. I selve rapporten illustreres og beskrives projektets vigtigste resultater, opdagelser og udviklingsprocesser. I dokumentationsbilagene findes yderligere detaljerede forklaringer, som der gennem hele rapporten bliver refereret til. Gruppens vejleder igennem projektforløbet har været Lars Mortensen.

2 Indledning

2.1 Problemformulering

På nuværende tidspunkt i Danmark lider mange af ludomani som i fleste tilfælde resulterer i et forringet arbejdsliv og familieliv. Ludomani foregår igennem spil, der går ud på at satse penge for at vinde en større gevinst end, hvad man satsede. De følgende gamblings spil kan være kasino, odds, gamblings-sider osv. Problematikken ligger i, at ludomaner bliver mindre bevidste over, hvor mange penge de spiller, hvilket medfører, at de ikke har råd til at dække deres dagligdagsbehov.

Det er en sygelig trang til at spille eller indgå i væddemål med mulighed for store gevinster. Undersøgelser fra ludomani.dk viser i 2017, at danskerne er begyndt at spille fra en yngre og yngre alder. Det viser en meget negativ udvikling i forhold til de tidligere år.

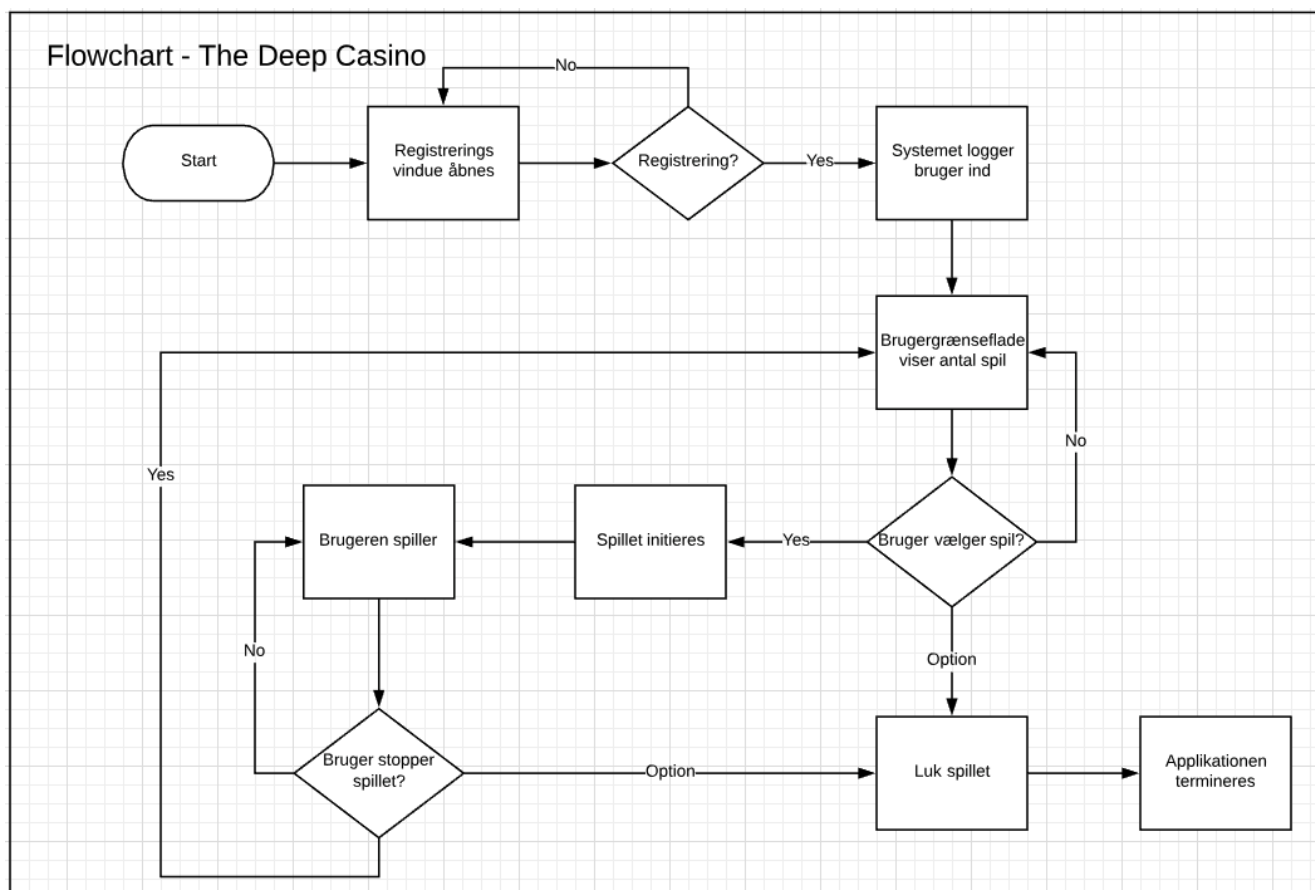
Med et virtuelt kasino med chance for stor gevinst uden at satse penge kan man som bruger få tilfredsstillet sine behov med at spille. Det fundamentale med det virtuelle kasino er, at man bruger virtuelle penge til at spille, på den måde undgås det store tab af sin formue.

2.1.1 Projektbeskrivelse

Konceptet går ud på at skabe et virtuelt Casino, hvor en bruger kan vinde et utal af præmier uden at bruge en eneste krone. Dette casino, The Deep Casino, giver mulighed for brugere at gamble med virtuelle penge som kan bruges i casinoets shop. Brugere af The Deep Casino behøver ikke nødvendigvis at vinde for at indtjene virtuelle penge. Den virtuelle valuta, som benyttes i The Deep Casino, kaldes deepcoins. Kasinoet forærer 100 deepcoins dagligt til de brugere, der logger ind. The Deep Casino vil samarbejde med villige sponsorater, som vil have boder i casinoets shop. Brugere af kasinoet har mulighed for at se reklamer fra vores sponsorater, og derved giver The Deep Casino 50 deepcoins til brugeren.

Med denne række af funktionaliteter ville The Deep Casino være en fyldestgørende erstatning til en gamblingplatform ift. de faktorer, som ludomaner generelt tilstræber gennem gambling med den store forskel, at de ikke har mulighed for at blive ruineret.

Måden, hvorpå at gruppen har tænkt sig, at det samlede system skal fungere, kan ses på figur 1.



Figur 1: Flowchart af The Deep Casino

2.2 Projektafgrænsning

Indenfor dette projekts grænser har det ikke været muligt at fuldende The Deep Casino. Gruppen har derfor udvalgt få UserStories til realisering baseret på de stillede krav til projektet og den givne tidsramme. I den forbindelse har gruppen opstillet en prioriteringsliste med projektets øverste mål.

2.2.0.1 Projektets øverste mål

- Udarbejde en prototype af The Deep Casino som en Windows-applikation.
- Applikationen skal kunne spille Blackjack og Roulette.
- Brugeren skal kunne vinde/tabe virtuelle penge i The Deep Casino.
- Applikationen skal have en database til at gemme brugerens oplysninger.

- Brugeren skal kunne logge ind med brugernavn.

Under udarbejdelse af dette projekt blev der i starten af semesteret drøftet nogle idéer for hvilke spil, der skulle være med i The Deep Casino. Gruppen blev enige om, at der maks skulle laves 3 spil til DeepCasino, men fandt derefter ud af, at det var for meget mht. tidsrammen. Gruppen har derfor valgt at udvikle blackjack og roulette.

Årsagen til at det netop blev blackjack og roulette, var fordi, at gruppen syntes, at blackjack og roulette var mest underholdende at arbejde med af de klassiske gambling-spil der findes.

3 Kravspecifikation

3.1 Funktionelle krav

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

| 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. | | |

3.2 Ikke-funktionelle krav

Der findes en række ikke-funktionelle krav, som kan ses i dokumentationen(2.2.1). Herunder stilles krav til The Deep Casino's opstillet efter FURPS.

3.3 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: 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.

4 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.

5 Metode og proces

Gruppen har anvendt SCRUM bl.a. i det, at opgaverne har været inddelt i sprints [Bilag 07]. Udviklingsforholdene har været således, at gruppen har opdelt sig i mindre sprint-grupper alt efter arbejdsfordelingen. Sprint-grupperne var ansvarlige for at de tildelte sprints blev løst. Under udviklingsprocessen forekom en masse frihed ift. planlægningen af sprints, da gruppen samtidig arbejder iterativt, så der er plads til at træde et skridt tilbage i arbejdet, når det er nødvendigt. På trods af at sprint-grupperne har fordybet sig i hver deres dele, har alle gruppemedlemmer i det hele taget været gode til at bevare et samlet overblik over det overordnede system, grundet gruppens hyppigt afholdte SCRUM-møder. Under disse møder blev de væsentlige SCRUM-elementer afdækket, da møderne resulterede i, at gruppen har fået snakket med hinanden om, hvad der er blevet lavet siden sidst, hvilke udfordringer der har været, og hvad der skal laves til næste møde. Det er vigtigt at få pointeret at gruppen ikke har anvendt SCRUM i punkt og prik i den klassiske forstand. Hvorledes gruppen har båret sig an med mht. det praktiske SCRUM er beskrevet i afsnittet Metode i dokumentationen.

Før gruppen for alvor begyndte at bruge SCRUM var alle gruppemedlemmer fælles om at udarbejde problemformuleringen, projektbeskrivelsen og kravspecifikationen for det overordnede produkt.

Under udarbejdelsen af projektet har gruppen anvendt en iterativ fremgangsmåde både i de komponenter, hvor gruppen har været fælles om afvikling og i de mindre sprint-grupper. Gruppen har fået gennemgået 2 reviews i løbet af projektet. Første gang fik gruppens kravspecifikation review af anden gruppe. Anden gang blev gruppens softwarearkitektur reviewet af gruppens vejleder, Lars Mortensen. Begge reviews har givet gruppen inputs til forbedringer af arbejdet, samtidig med at gruppen har fået et andet perspektiv på semesterprojektet.

6 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.

6.1 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, hvilket også er taget udgangspunkt i user story 2. 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).

6.2 Blackjack

Til udførelsen af BlackJack-applikationen ift. User story 3, har gruppen haft en del overvejelser i forbindelse med at gøre brug af SOLID-designprincipperne og design patterns. Gruppen er ude efter et design pattern, der er oplagt at vælge ift. effektivisering af applikationer, der gør brug af grafisk brugergrænseflader. Gruppen har derfor overvejet tre forskellige design patterns - nemlig MVC(Model-View-Controller), MVP(Model-View-Presenter) og MVVM(Model-View-ViewModel). Disse tre design patterns er alle under kategorien GUI patterns.

Grunden til at gruppen har valgt at anvende MVVM design pattern er, at 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 ift. test af applikationen via unit test. Ydermere vil brugen af MVVM give os ensartet kodestruktur, at det kan implementeres

i samtlige øvrige applikationer, der evt. kunne indgå i gruppens solution, da det ikke er begrænset til WPF-applikationer. MVVM pattern skal anvendes i BlackJack-applikationen til primært at adskille business logic(Model og ViewModel) fra presentation logic(View). Dette design pattern giver også generelt meget løs kobling. Den nævnte adskillelse og den løse kobling i dette design pattern giver mulighed for flere udviklere at arbejde samtidigt på projektet, i tilfælde af at den ene arbejder med presentation logic og den anden med business logic. Årsagerne til valget af MVVM samt dybere forklaring på nævnte fordele og en potentiel ulempe ved MVVM er beskrevet i dokumentationen (5.2.1).

For at komme potentielle softwarekvalitetsproblemer i forkøbet og opnå optimal og velfungerende kode der kan sikre kvalitetssikret software, er statisk analyse en mulig løsning. Statisk analyse giver et overblik over disse problemer og giver gruppen mulighed for at eksaminere kode uden at køre programmet. Der er ting, som statisk analyse ikke kan identificere, eksempelvis kan statisk analyse ikke registrere om, eventuelle softwarekrav er opfyldte, eller hvordan en funktion vil eksekveres - her er dynamisk analyse bedre. Statisk analyse opdager fejl i kode tidligt. Det fremskynder udviklingen ved at sikre, at testprocesserne er mere effektive, hvorfor gruppen i sidste ende vælger statisk analyse. En mere konkret beskrivelse og forklaring af statisk analyse findes i "Static analysis og software matrice" under Afsnittet(5.2.2) i dokumentationen.

6.3 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. Gruppen fandt det vigtigt at opfylde de mest betydende User Stories for roulette-applikationen, for at gøre spillet så realistisk som muligt for brugeren. Derfra satte gruppen sig ind i, hvordan et moderne roulette spil fungerede, og hvilke funktionaliteter det havde.

Den første tanke, der strejfede gruppen, var, at koden skulle skrives i sproget C#, da dette var det grundlæggende sprog for dette semester. 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.

6.4 Opstartsapplikation

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 bør 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 dokumentation. Gruppens forestilling omkring opstartsapplikationen har været at holde applikationen overskuelig og simpel.

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 et program, der kan skifte mellem sider/vinduer, således at man som bruger ikke har mere end ét vindue åbent.

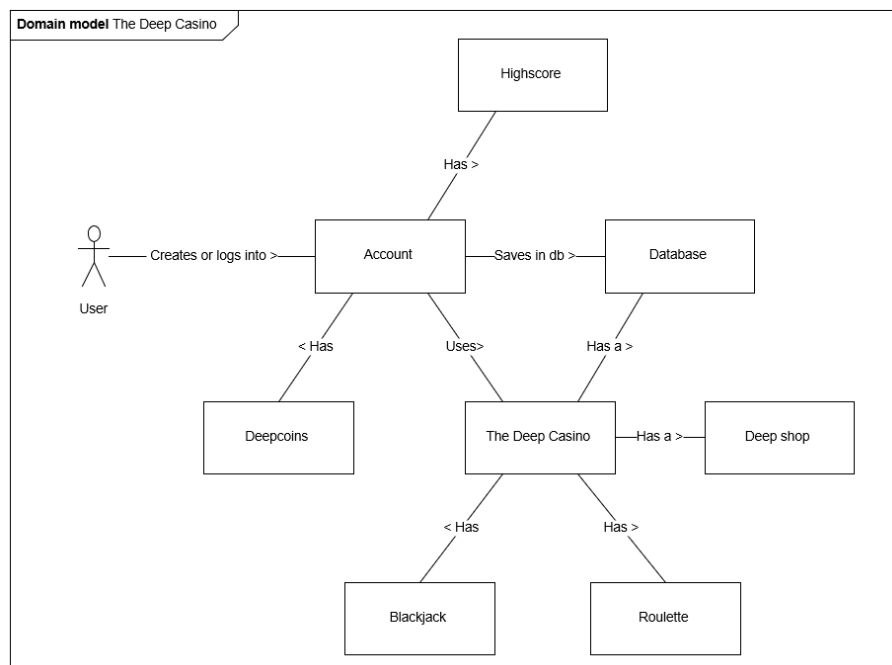
En essentiel model, som gruppen tænker at anvende i forbindelse med nævnte funktionalitet i denne del, er modellen ContentControl. Denne model høre under et bredt emne inden for GUI kaldet The Content Model. ContentControl er en model, som bruges ift. funktionalitet i en WPF-applikation. Mere specifikt 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.

7 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.

7.1 Domænemodel

På figur 2 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 2: Domænemodel af systemet

7.2 Applikationsmodel

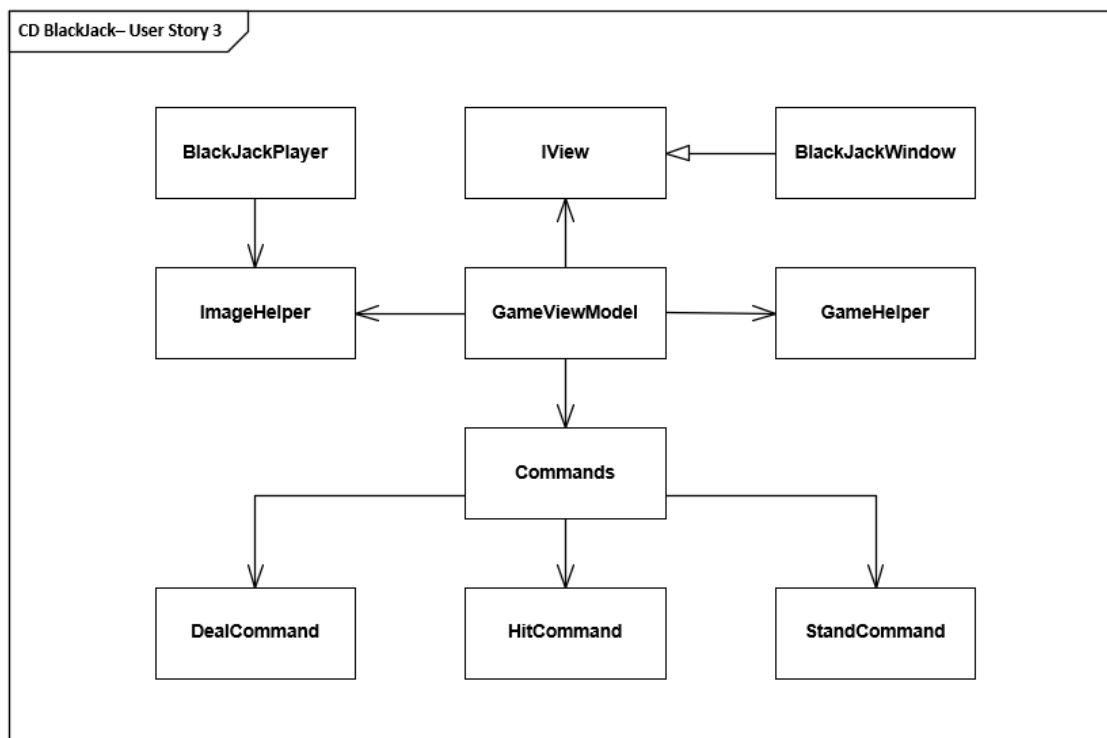
Med et indblik på domænemodellen er det næste aktionspunkt at udarbejde applikationsmodellen. Tidligere blev der nævnt at lave et applikationsmodel på de tre essentielle User Stories for The Deep Casino. Indledningsvist er der lavet generelle klassediagrammer for User Story 3, User Story 4 og User Story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

7.2.1 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 3 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.

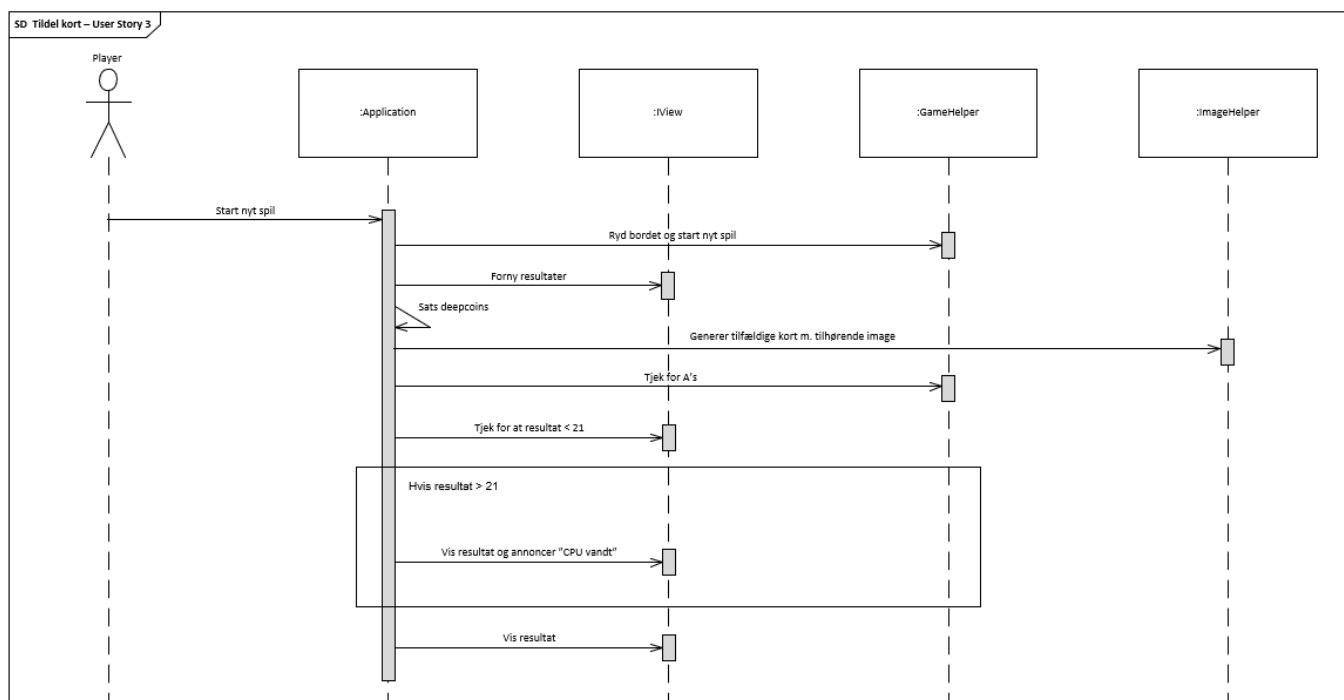
7.2.1.1 Klassediagram



Figur 3: Klassediagram for hele BlackJack-applikationen

På sekvensdiagrammet på figur 4 illustrerer gruppen, hvorledes kommunikationen mellem klasserne i store træk kommer til at foregå. Klassen Application er en sammensætning af GameViewModel, Commands og de klasser, som Commands bruger (fra klassediagrammet på figur 3).

7.2.1.2 Sekvensdiagram



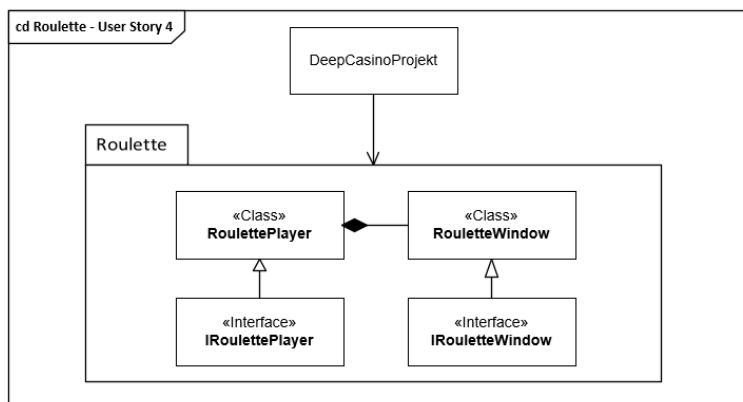
Figur 4: Sekvensdiagram for start af BlackJack

7.2.2 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.

7.2.2.1 Klassediagram

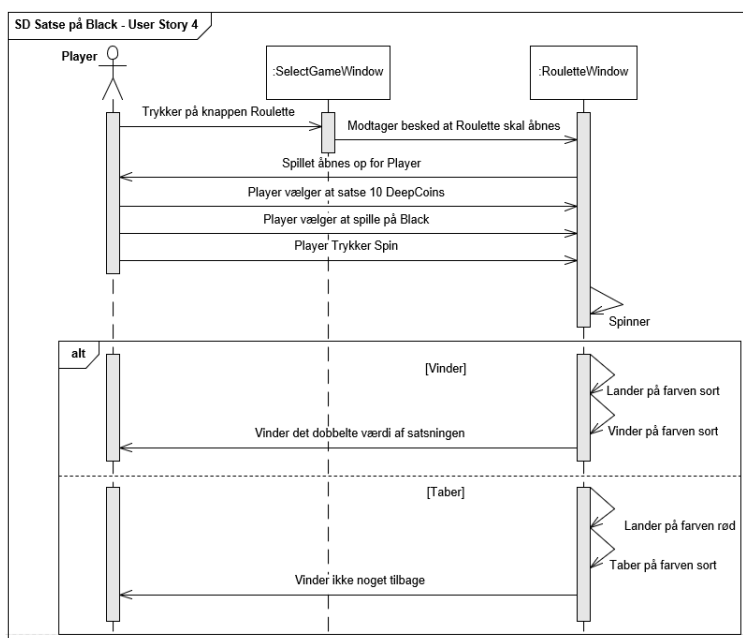
Det ses på figur 5 nedenfor, hvordan samarbejdet mellem klasserne er opstillet. Pilene viser, hvordan blokkene er associeret med hinanden. I skitsen kaldet "Roulette" ses fire blokke. Disse skal ses som applikationen for spillet roulette. Det ses også, at en blok kaldet DeepCasinoProjekt er associeret med skitsen Roulette, denne blok er Opstartsapplikationen, som ses på figur 7 i underafsnittet User Story 5 - Opstartsapplikation. Overordnet viser diagrammet kommunikationen af Roulette-applikationen.



Figur 5: Klassediagram for hele Roulette-applikationen

7.2.2.2 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 6 nedenfor viser en sekvens af en spiller, der satser på farven sort i Roulette-applikationen.



Figur 6: Sekvensdiagram for satsning af farven sort på Roulette-applikationen

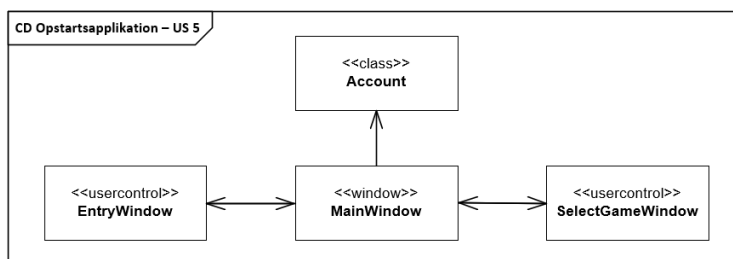
Til start trykker Player på knappen Roulette, som vil starte spillet roulette. Player vælger at satse ti deepcoins på farven sort. Dernæst trykker Player på knappen Spin, og spillet kører. Afhængig af hvad resultatet vil blive, kan Player enten vinde eller tabe. Hvis det er tilfældet, at der landes på 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.

7.2.3 User story 5 - Opstartsapplikation

Dette afsnit indeholder diagrammer over gruppens opstartsapplikation. I forhold til gruppens domænemodel på figur 2 svarer opstartsapplikationen til "The Deep Casino-blokken.

7.2.3.1 Klassediagram

På figur 7 illustreres, hvordan gruppen bygger og sammensætter de forskellige komponenter af opstartsapplikationen samt klassen Account.

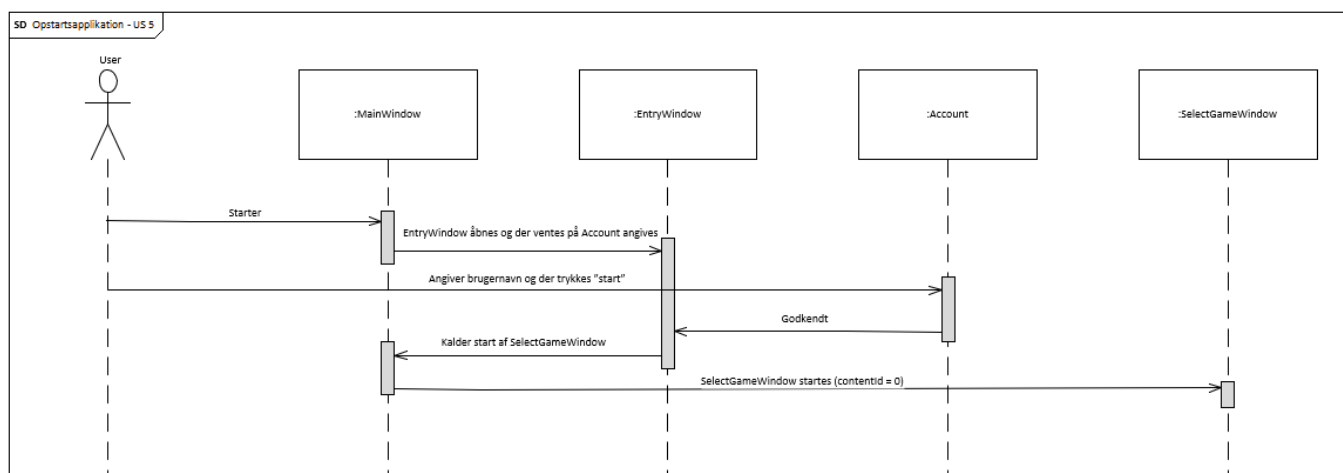


Figur 7: Klassediagram for opstartsapplikationen.

Account fra figur 7 er en klasse i applikationen, hvori brugerens informationer skal gemmes. MainWindow skal køres så snart, at applikationen startes og giver ikke i sig selv noget visuelt billede for brugeren, men den sørger for at EntryWindow aktiveres. EntryWindow er vinduet, som bliver åbnet så snart, at applikationen køres. Gennem EntryWindow skal brugeren logge ind. SelectGameWindow aktiveres efter, at brugeren er logget ind gennem EntryWindow, og deri kan brugeren vælge det spil, der skal åbnes.

7.2.3.2 Sekvensdiagram

På figur 8 ses et sekvensdiagram til illustrering af, hvorledes opstartsapplikationen køre fra, at brugeren starter applikationen til, at han/hun når til SelectGameWindow.



Figur 8: Sekvensdiagram over opstart af The Deep Casino-applikationen.

8 Softwaredesign

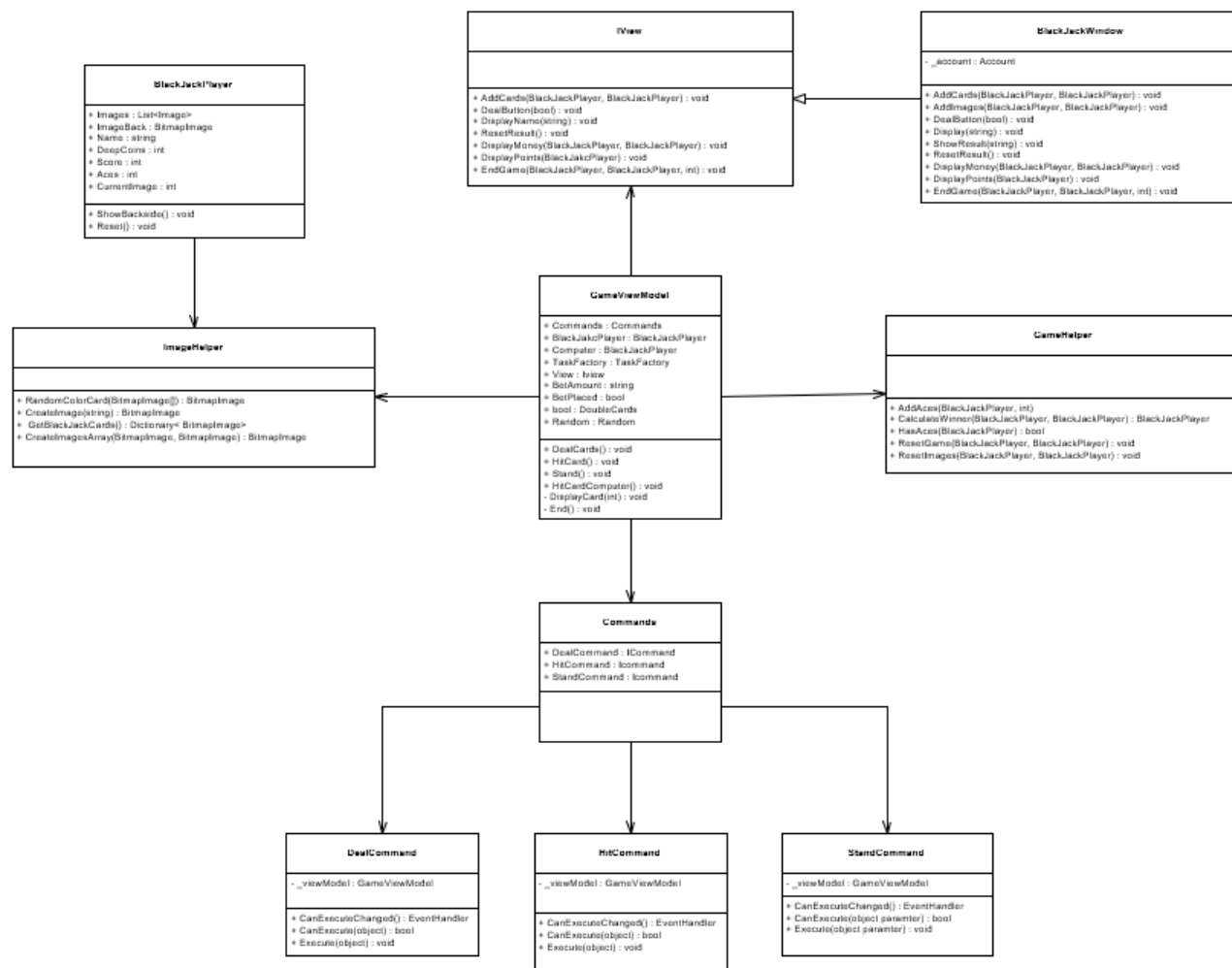
Der er i projektgruppen blevet arbejdet med at få The Deep Casinos funktionaliteter på plads. Følgende punkter forneden beskriver henholdsvis softwaredesignet for Databasen, opstartsapplikationen og implementering af de to Casino-spil; BlackJack og Roulette.

8.1 BlackJack

Her beskrives den interne funktionalitet af klasserne/vinduerne, som gruppen har valgt at arbejde med for at fyldestgøre US 3. Derfor har gruppen i dette afsnit inkluderet et mere detaljeret klasse- og sekvensdiagram med attributter og funktionskald.

8.1.1 Klassediagram

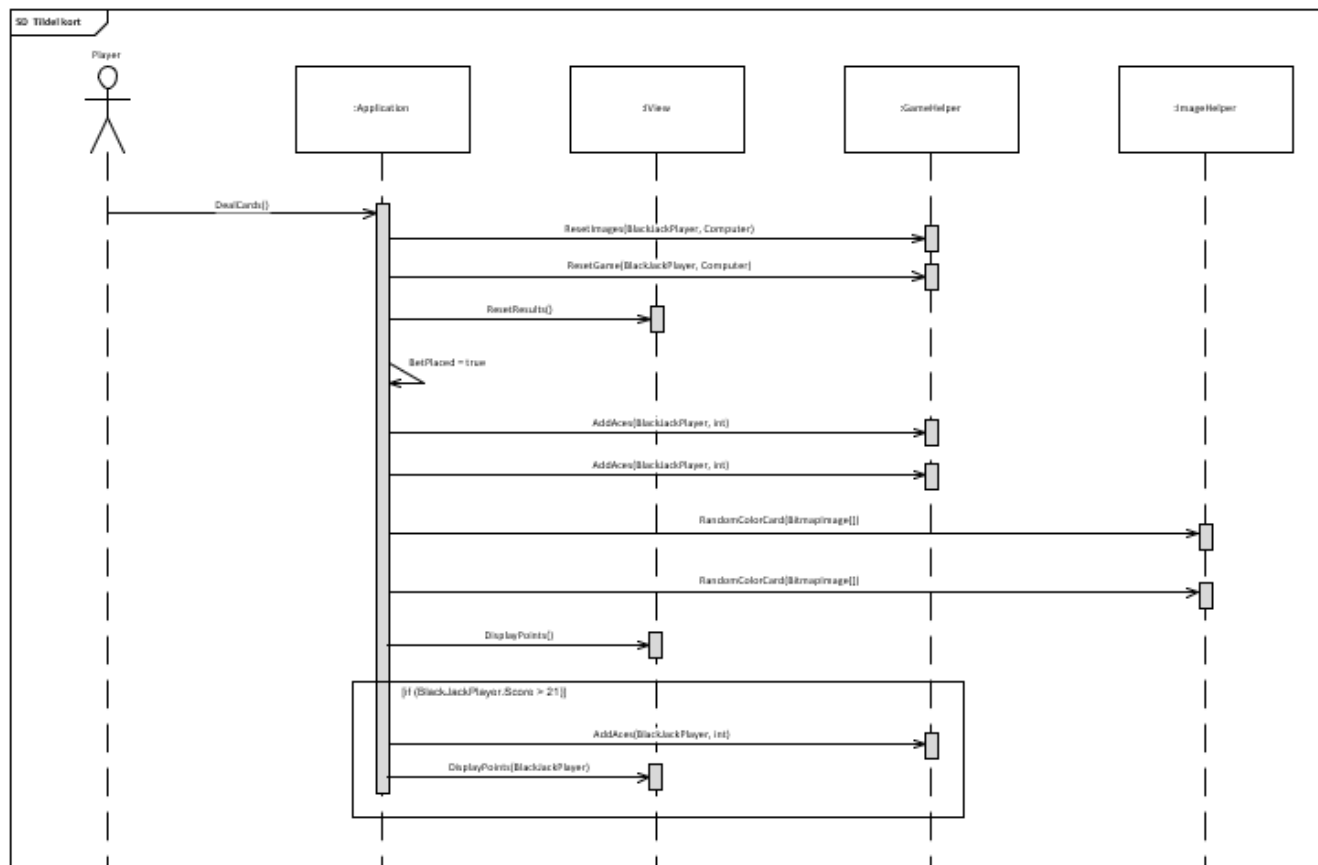
På figur 9 ses det fulde klassediagram for BlackJack med alle klasser og tilhørende attributter. Disse klasser er alle inkluderet i BlackJack-applikationen og en del af en nødvendig helhed for at opfylde US 3. Klassernes formål og anvendelse har gruppen beskrevet implementeringsafsnittet under BlackJack.



Figur 9: Klassesdiagram for hele BlackJack-applikationen

8.1.2 Sekvensdiagram

Dette sekvensdiagram(figur 10) tilhører klassesdiagrammet på figur 9. Sekvensdiagrammet her viser forløbet "Startspil". Startspil betegner det forløb, hvor brugeren allerede har åbnet BlackJack-applikationen og derefter starter et nyt spil. Sekvensdiagrammet giver et indblik i hvorfra, at de forskellige metoder kaldes og i hvilken rækkefølge.



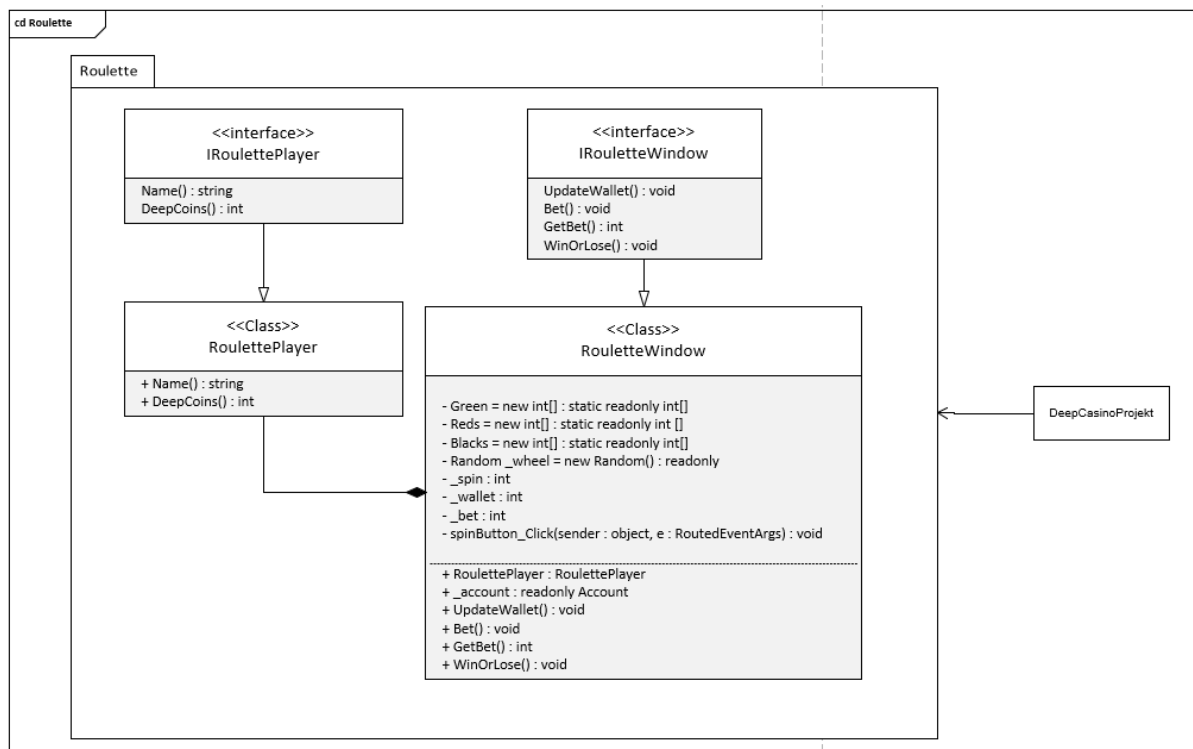
Figur 10: Sekvensdiagram for start af BlackJack.

8.2 Roulette

Dette underafsnit har til formål at beskrive funktionaliteten af Roulette. Der vil fremgå et udarbejdet klassediagram med attributter og et sekvensdiagram med funktionskald baseret på de hidtidige diagrammer fra afsnittet Softwarearkitektur.

8.2.1 Klassediagram

På figur 11 nedenfor ses det udarbejdet klassediagram til roulette, hvor attributter er inkluderet. Diagrammet danner funktionaliteten, der opfylder kravene til scenariet for sekvensdiagrammet. Diagrammet viser det overordnede kommunikationen mellem klasserne for både Roulette-applikationen og Opstarts-applikationen.

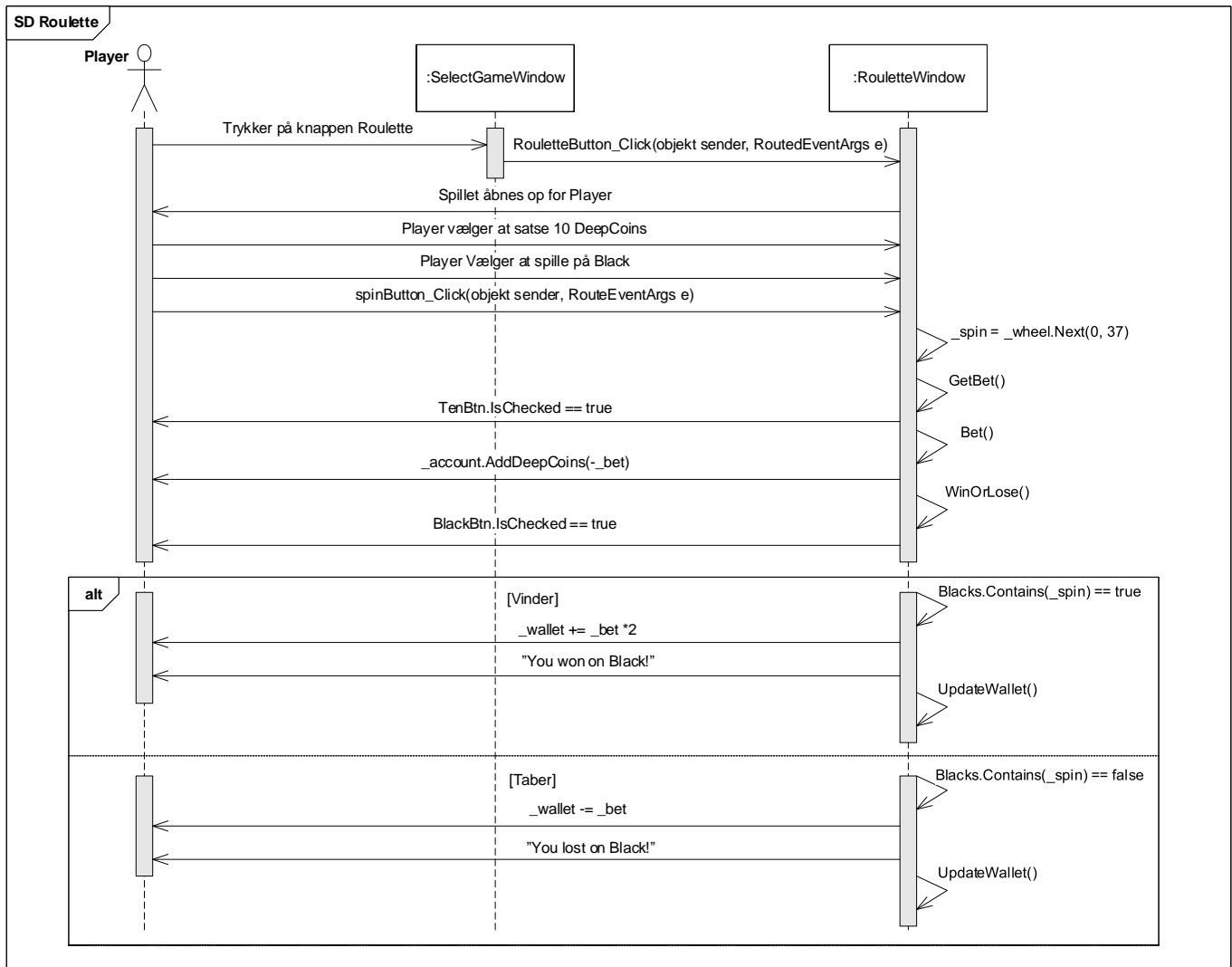


Figur 11: Klassediagram for Roulette-applikationen

8.2.2 Sekvensdiagram

Eftersom attributterne er fundet, er der udarbejdet et sekvensdiagram, som ses på figur 12 nedenfor. Diagrammet illustrerer et af scenarierne for Roulette-applikationen, da de er ensartet. Dette scenarie omhandler spilleren, der satser ti deepcoins på farven sort.

En mere uddybende beskrivelse af dette sekvensdiagram kan læses mere om i dokumentationen under afsnittet Softwaredesign.



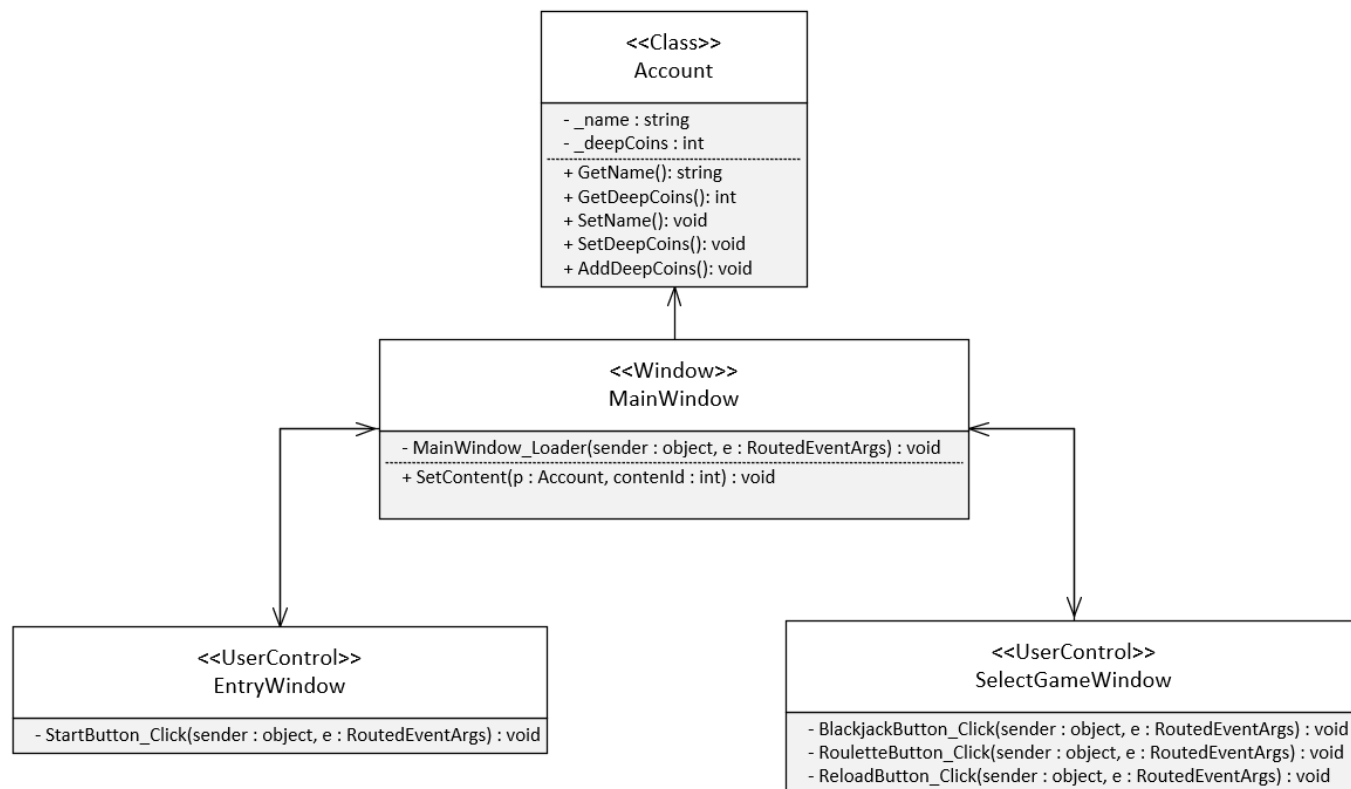
Figur 12: Overflødigt sekvensdiagram af Roulette - Satse ti deepcoins på farven sort

8.3 Opstartsapplikation

Her beskrives den interne funktionalitet i blokkene, som er beskrevet under "Opstartsapplikation" i afsnittet softwarearkitektur. Med henblik på at få løst US 5 beskriver gruppen her det overordnede system for opstartsapplikation. I den sammenhæng er der også inkluderet et mere detaljeret klasse- og sekvensdiagram med attributter og funktionskald i forhold til de tidligere viste i afsnittet Softwarearkitektur.

8.3.1 Klassediagram

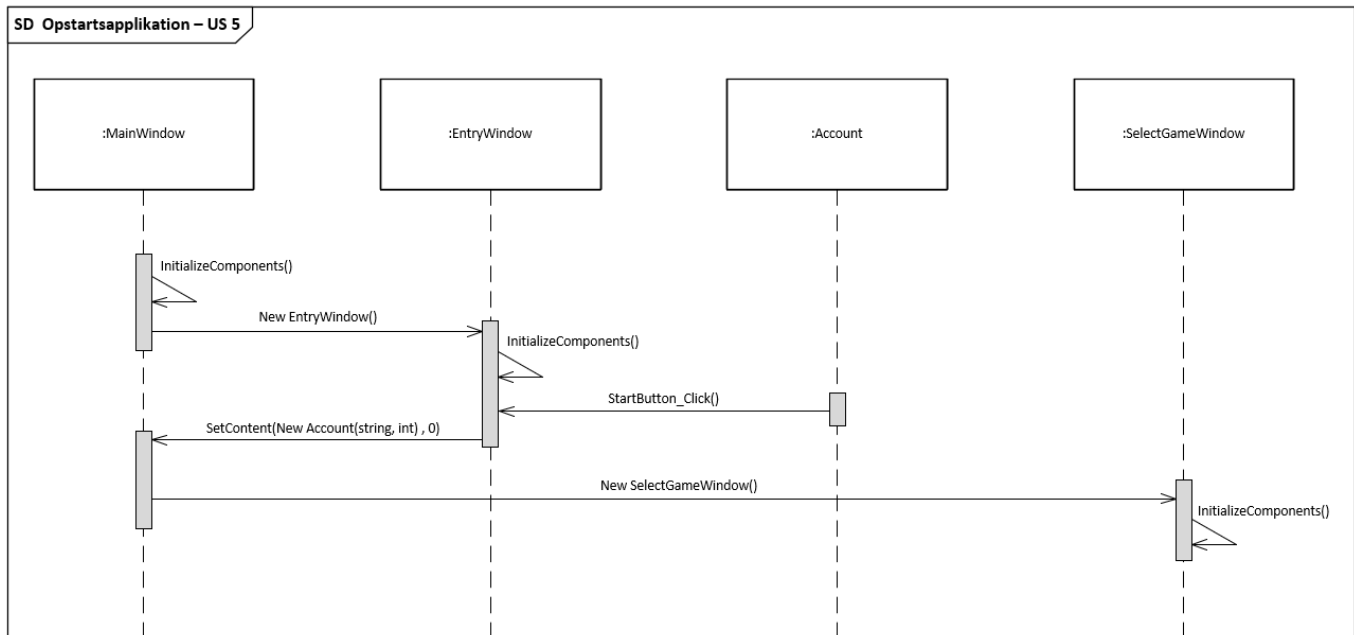
På figur 13 ses klassediagrammet over opstartsapplikationen. De forskellige blokke er kort beskrevet i afsnittet softwarearkitektur, dog ses tilhørende attributter og metoder her under hver enkel blok.



Figur 13: Klassediagram for opstartsapplikationen.

8.3.2 Sekvensdiagram

På sekvensdiagrammet(figur 14) illustreres, hvornår metoderne fra klassediagrammet(figur 13) bliver kaldt og i hvilken rækkefølge. Yderligere beskrivelse af metodernes algoritme og funktion finder sted i implementeringsafsnittet. En mindre gennemgang af trinnene i diagrammet kan findes i dokumentationen(7.3.2).



Figur 14: Sekvensdiagram for opstartssaplikationen.

9 Software implementering

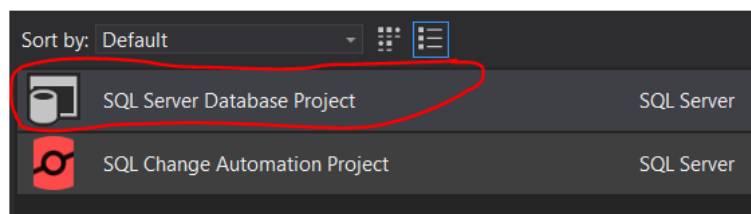
Under dette punkt vil der blive gennemgået, hvordan udarbejdningen af de forskellige arbejdsfordelinger af The Deep Casino er blevet implementeret.

9.1 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. I dette afsnit vil der være fokus på Create operationen. De andre operationer er beskrevet i dokumentationen(8.1.1).

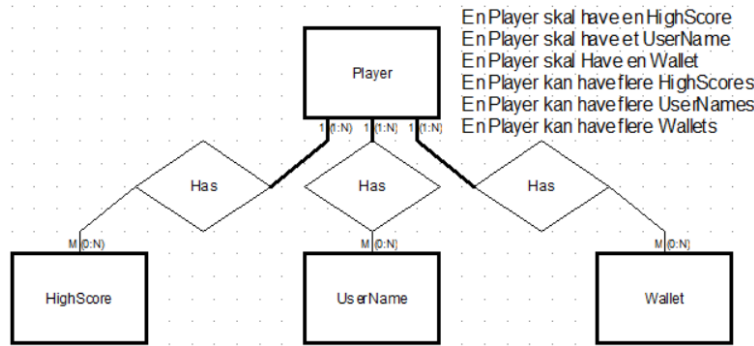
Den Relationelle database er arbejdet ud fra et logisk skema fra et program, DDS-Lite.

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



Figur 15: Oprettelse af SQL Server Database

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



Figur 16: Entity Relationship Diagram for Databasen

Ud fra ERD bliver der automatisk bygget en Crow's Foot diagram. Yderligere detaljer kan læses på dokumentationen(8.1).

9.1.1 CRUD operationer

Der skal nu laves CRUD operationer i en .cs fil. Bemærk at Create operationen er den der bliver beskrevet her. De andre operationer kan findes i dokumentationen(8.1.1). Der oprettes en cs-fil DeepCasinoDBUtil.cs, hvilket er her, Create operationen 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=
6         E18I4DABau556770;User ID=E18I4DABau556770;Password=E18I4DABau556770;Connect Timeout=30;Encrypt
7         =False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False");
8         con.Open();
9         return con;
10    }
11 }

```

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 Create operationen for entiteten, Player. Create operationen bliver også kaldt for "Add".

```

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 parameteren, 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, skal der nu laves en applikation, der kalder på den. Gruppen har valgt at kalde denne fil for `DCAApp`, forkortet `DeepCasinoApp`.

```

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);
10        util.DeletePlayerDB(ref newPlayer);
11        util.UpdatePlayerDB(ref newPlayer);
12        util.GetPlayerDB();
13    }
14 }

```

Listing 3: DCAApp hvor der bliver kaldt på Create operationen

Som man kan se på Listing 3, 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. På linje 8 er der, hvor vi kalder på Create operationen, der er blevet oprettet. Man ser også, at der er en reference til `Player` objektet med de specifikke værdier. 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 4: `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!

9.1.2 Forbindelseleddet mellem applikationen og Database

Under dette punkt ses implementering af forbindelseleddet mellem databasen og applikationen.

Der er benyttet `SqlConnection` til at forbinde til databasen via. en connection string, hvor der herefter har benyttet `SqlCommand` til at overføre værdierne til tabellen i databasen.

```

1 private void StartButton_Click(object sender, RoutedEventArgs e)
2 {
3     string name = lestdoit.Text;
4     MainWindow window = (MainWindow)Window.GetWindow(this);
5     window.SetContent(new Account(name, 1000), 0);
6     var con = new SqlConnection(@"Data Source=st-i4dab.uni.au.dk;Initial Catalog=E18I4DABau556770;
7     User ID=E18I4DABau556770;Password=E18I4DABau556770;Connect Timeout=60;Encrypt=False;
8     TrustServerCertificate=True;ApplicationIntent=ReadWrite;MultiSubnetFailover=False");
9     con.Open();
10    var query = "INSERT Player (HighScore, UserName, Wallet) VALUES (0,'" + this.lestdoit.Text +
11    "',1000)";
12    SqlCommand cmd = new SqlCommand(query, con);

```

```
10 cmd.ExecuteNonQuery();
11 con.Close();
12 }
```

Listing 5: StartButton_Click() - funktionen til at oprette forbindelse til databasen

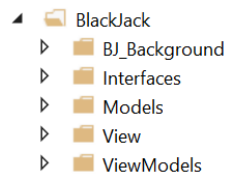
9.2 BlackJack

I dette afsnit har gruppen kort beskrevet de overvejelser, der har været under implementeringen af den endelige BlackJack-applikation mht. SOLID-principperne og MVVM. Selve implementeringen er naturligvis også beskrevet herunder.

Applikationen bærer til dels præg af, at gruppen har haft SRP og DIP i baghovedet under udarbejdelsen. SRP 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. DIP ses ved forholdene mellem GameViewModel, IView og BlackJackWindow. Dette uddybes i dokumentationen(8.2.1).

I dokumentationen(8.2.2) går gruppen i dybden med implementeringen MVVM samt de faktorer, der spiller en væsentlig rolle, for at gøre det muligt at bruge design pattern i applikationen. Ligeledes gøre gruppen brug af kodeudsnit til at illustrere, hvordan data binding er anvendt i forbindelse med interfacet ICommand.

Måden, hvorpå gruppen har grebet MVVM design pattern an på, var at starte med at inddеле de tre overordnede lag, som er beskrevet i analysen.



Figur 17: 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-mappen finder vi BlackJackWindow.xaml samt BlackJackWindow.xaml.cs, som har til formål at indkapsle user interface. Det er denne del af applikationen, som brugeren interagerer med og visuelt kan se.

BlackJackWindow skal have kendskab til BlackJackViewModel, hvilket bliver fuldført i constructoren for det bagvedliggende kode for BlackJackWindow. Fremgangsmåden er at sætte DataContext til at være lig med BlackJack-ViewModel.

```
1 public BlackJackWindow()
2 {
3     InitializeComponent();
4     DataContext = new GameViewModel(this, __account);
```

```
5      }
```

Listing 6: BlackJackWindow

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 7: AddImages

Det ses oftest, at disse commands bliver behandlet i viewmodel (BlackJackViewModel), men gruppen har for overskuelighedens skyldt valgt at implementere 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 tage udgangspunkt i DealCommand:

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

Listing 8: 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 sat 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 9: 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 alt afhængigt af, om brugeren har indtastet den rigtige værdi, hvilket vil sige, at CanExecute ikke bliver 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 10: 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 implementering af ICommand.

9.2.1 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:

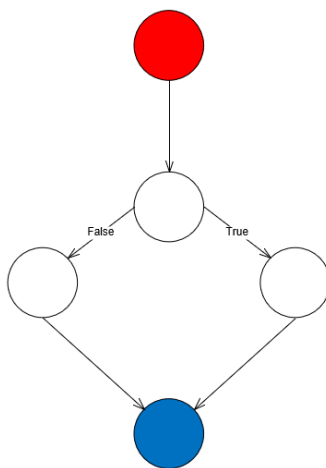
```
1    public void DisplayPoints(BlackJackPlayer player)
2    {
```

```

3      if (player.Name == "Computer")
4      {
5          ComputerPoints.Content = player.Score;
6      }
7      else
8      {
9          PlayerPoints.Content = player.Score;
10     }
11 }

```

Listing 11: DisplayPoints



Figur 18: 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 lavet to test cases for at få fuld covarge på denne funktion. Det udregnede resultatet af Cyclomatic Complexity stemmer ligeledes i overensstemmelse med kode 11. Koden er bestående af en if/else statement, hvilket vil sige, at der kan være to resultater. Det vil her være oplagt at anvende unittest for at reelt kunne dokumentere udkommet af koden.

9.3 Roulette

Dette afsnit har til formål at vise implementeringen af Roulette-applikationen, hvor der indgår en gennemgang af den interne funktionalitet. Der vil indgå kodeudsnit og beskrivelser på de mest betydende funktioner, inklusiv

private objekter, der reflekterer fundamentet af applikationen. De resterende kan findes i dokumentationen under afsnittet Software implementering.

Med udgangspunkt i scenariet fra sekvensdiagrammet er der udvalgt de mest relevante funktioner derfra.

9.3.1 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 12: 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 fire andre objekter, der har til formål at blive anvendt på forskellige måder. Disse vil være beskrevet løbende for de senere funktioner heri.

9.3.2 spinButton_Click()

Denne funktion har til formål at spinde 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 13: spinButton_Click()

Det ses på det ovenstående kodeudsnit, at funktionen starter med at generere et tilfældigt nummer fra 0 til 37, som den tilsætter til det private objekt, `_spin`. 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 genererede nummer. Til sidst kaldes funktionen `UpdateWallet()`, som anvendes for at sikre, at pungen viser det oprindelige værdi i applikationen.

9.3.3 Bet()

`Bet()` funktionen beskæftiger sig med satsningen for applikationen. Funktionen anvender sig af `_bet`, som kan ses på ovenstående kodeudsnit, Private properties. Derudover 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 sikrer, 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 14: Bet()

9.3.4 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    }
```

```
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 15: WinOrLose()

9.4 Opstartsapplikation

Her kommer gruppen ind på opstartsapplikationen, hvor en gennemgang af metoderne i de brugte klasser og vinduer gennemgås. 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 16: 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 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.

For at kunne realisere ovennævnte funktion, gør gruppen brug af en ContentControl, som gruppen har instantieret som ContentHolder. Dette er yderligere beskrevet i dokumentationen(8.4).

Følgende metode findes 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 17: Metoden BlackjackButton_Click

Denne metode kaldes så snart, at brugeren fra SelectGameWindow trykker på knappen "BlackJack". Det mest bemærkelsesværdige er, at contentId i SetContent-funktionen 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.

10 Test

Dette punkt beskriver den fremgangsmåde, som er valgt for at teste de forskellige operationer i DeepCasino samt hvilke resultater, der forventes af disse tests.

10.1 Database - Test

Dette punkt beskriver de unit tests, der foretages af databasen for at se, om de implementerede funktionaliteter opfører sig, som forventet. Unit testene for databasen bliver taget udgangspunkt i User story 2. Det forventes, at de implementerede operationer virker efter hensigten. Disse tests består af fire grundlæggende operationer, der har hver sine funktioner til at kunne udføre en bestemt handling. Operationerne skal bl.a. kunne oprette en ny Player i databasen, og der skal være en række af informationer tilknyttet til denne Player. De kommende afsnit beskriver disse CRUD-operationer.

10.1.1 CRUD operationer - Test

De implementerede CRUD-operationer for Player skal testes under dette punkt. Der er fire grundlæggende CRUD-operationer, hvor hver af disse har sin metode til at gøre noget bestemt. De implementerede CRUD-operationer er: Create, Read, Update og Delete.

Den første CRUD-operation, der skal testes er Create. Create-operationen skal kunne oprette en ny Player i databasen, og der skal være tilhørende værdier tilknyttet til denne nye Player.

10.1.1.1 Create - Test

Dette punkt gennemgår test for Create-operationen, hvor der testes efter, om operationen virker efter hensigten. Det er en forudsætning, at der skal være en tilgængelig database, før operationen kan fungere. Programmet bliver eksekveret med en tilgængelig database, og værdierne angives i DCApp.cs, og der kan herefter tjekkes efter i databasens tabel, om de indtastede værdier kommer ind og stå i databasen. En mere dybdegående forklaring af testen kan findes i Dokumentation.

Nu da Create-operationen er testet, skal de indtastede oplysninger kunne hentes med en get-metode, til dette er det blevet implementeret en Read-operation, som vil blive gennemgået i næste punkt.

10.1.1.2 Read - Test

Under dette punkt skal Read-operationen testes. Det er en forudsætning, at der skal være indtastede oplysninger i databasen, før Read-operationen kan testes. Read-operationen vil fungere uanset, om der er indtastet oplysninger i databasen eller ej, men for at teste, at operationen faktisk virker, kræves det, at der findes data i databasen. Programmet eksekveres, og et terminalvindue vil dukke op, hvor indholdet af data fra databasen vil blive udskrevet. En mere dybdegående forklaring af testen kan findes i Dokumentation.

Når Read-operationen er testet, skal der testes efter, om de indtastede oplysninger kan ændres. For at kunne ændre indtastede værdier i databasen er det blevet implementeret en Update-operation, som har det formål, at det skal kunne ændre data, som er gemt i databasen. Næste punkt vil beskrive den test, der udføres for Update-operationen.

10.1.1.3 Update - Test

Dette punkt beskriver fremgangsmåden for den test, der bliver udført på Update-operationen. Det er en forudsætning, at en Player er oprettet og findes i databasen, før Update-operationen kan anvendes. Når en Player er opdateret, skal der tjekkes efter i tabellerne i databasen, om de værdier stemmer overens med de nye indtastede oplysninger. En mere dybdegående forklaring af testen kan findes i Dokumentationen.

Når Update-operationen også er testet, skal den sidste af de fire CRUD-operationer testes, som er Delete-operationen. Næste punkt vil gennemgå de tests, der udføres for Delete-operationen.

10.1.1.4 Delete - Test

Dette punkt gennemgår de tests, der udføres for Delete-operationen. Det er en forudsætning, at der skal være en eksisterende Player i databasen, før operationen kan fungere. Delete-operation har til formål, at den går ind og fjerner en Player fra databasen og dens tilhørende værdier. Når en Player er slettet, skal der tjekkes efter i tabellerne, som findes i databasen, om den ønskede Player faktisk er slettet fra databasen. En mere dybdegående forklaring af testen kan findes i Dokumentationen.

Når de fire CRUD-operationer nu er testet, skal forbindelsen mellem databasen og DeepCasino testes. Det næste punkt gennemgår de tests, som udføres for forbindelsen.

10.1.2 Forbindelsesledet mellem databasen og DeepCasino - Test

Under dette punkt bliver der testet forbindelse mellem databasen og DeepCasino, ved at der sker en event på applikationen. I det øjeblik brugeren indtaster et navn og trykker på start-knappen, sker der en event. Dvs. dataene bliver sendt ved at der oprettes en SQL-forbindelse til det lokale-database ((localdb)\MSSQLLocalDB), hvor dataene bliver overført ved at benytte SQLCommand. Brugerens navn og Deepcoins vil blive gemt på tabellen i databasen. Når dataene er gemt, skal der testes forbindelse til SQL-server(st-i4dab.uni.au.dk), i stedet for lokale-database. SQL-server(st-i4dab.uni.au.dk) er oprettet af underviseren fra I4DAB-kurset. Gruppen har valgt at der skal oprettes forbindelse til denne server, så man er i stand til at kunne modtage dataene fra en anden enhed.

Nu når Forbindelsesledet mellem databasen og DeepCasino er testet, skal BlackJack-applikationen testes. BlackJack-applikationen har en række funktionaliteter, som giver brugeren forskellige muligheder. Næste punkt vil gå i dybden med de tests, som udføres for BlackJack-applikationen.

10.2 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 er i overensstemmelse med accepttest.

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 det her muligt, at se at brugeren får tildelt det navn som bliver indtastet i opstartsapplikationen, hvilket kan ses, ved at der sættes et 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.

10.3 Roulette - Test

Der vil i dette afsnit blive givet en gennemgang af de forskellige test som er udført på roulette-applikationen. Testene vil tage udgangspunkt i nogle af de angivne User Stories for applikationen.

Testen tager udgangspunkt i User Story 4 og User Story 1, hvor målet er at illustrere, at en spiller får givet et navn og tildelt sig nogle deepcoins inde fra opstartsapplikationen. Igennem Visual Studio debuggeren, testes de enkelte dele af applikationen igennem, for at tjekke om spillerens deepcoins bliver anvendt som antaget igennem roulette-applikationen.

Den visuelle test af applikationen er foretaget igennem accepttesten, hvor den interne funktionalitet vil blive udført under afsnittet for Resultater. Testen bliver udført med debuggeren, for at sikre sig at de forskellige attributter indeholder de rigtige parametre afhængig af, hvad det er der foretages på applikationen.

Der er i forbindelse med testningen ikke blevet foretaget unit Test og integrationstest, hvilket også foreligger under afsnittet Diskussion.

10.4 Opstartsapplikation - Test

Med udgangspunkt i implementeringen 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 der 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 er der skal testes er tilsvarende som det første, dog med roulette-vinduet i stedet for. Begge dele testes visuelt ved kørsel af applikationen. Er det tilfældet at vinduet skiftes, består funktionaliteten testen.

11 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 |

11.1 Database

Dette afsnit vil komme med nogle eksempler på resultater, der fremkommer under udførsel af tests af de fire CRUD-operationer, som indgår i databasen. De fire operationer, som resultaterne fremkommer af, er følgende: Create, Read, Update og Delete.

11.1.1 Create

Dette afsnit går i dybden med de resultater, der fremkommer under tests af Create-operationen. Der er blevet implementeret noget kode, som har til formål at oprette en ny Player, som bliver gemt i databasen.

Når programmet køres, dukker et terminalvindue op, der meddeler, at en ny Player er blevet tilføjet, og er nu i databasen.

```

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

Figur 19: Terminalvindue udskrift

Efter at meddelelsen er modtaget, skal der tjekkes efter i tabellerne, som tilhører databasen. Her kan det bekræftes, at en ny Player faktisk er blevet oprettet. På billedet nedenunder kan det ses.

| | PlayerID | HighScore | UserName | Wallet |
|--|----------|-----------|----------|--------|
| | 8 | 1000 | Foo | 100 |

Figur 20: Tabeloversigt for Databasen

Det kan nu bekræftes, at en ny Player er oprettet og er i databasen med de tilknyttede værdier. En mere dybdegående forklaring af resultaterne kan findes i Dokumentation. Da det nu kan konkluderes, at Create-operationen fungerer efter hensigten, skal man kunne tilgå disse data med en get-metode. Næste afsnit vil komme ind på de tests, der bliver udført for Read-operationen, som har til formål at udskrive de informationer, som er gemt i databasen.

11.1.2 Read

Dette afsnit vil komme ind på de resultater, som fremkommer under de tests, der udføres på Read-operationen. Read-operationen fungerer som en get-metode. Der er implementeret noget kode, som udskriver alt det, som findes i databasen, ud på terminalvinduet. De informationer, som udskrives på terminalvinduet, er brugernavn (UserName), score (HighScore) og pung (Wallet). For at teste at metoden faktisk virker, blev der arbejdet videre med dataene fra forrige afsnit. Programmet bliver kørt, 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 21: Terminaludskrift af Read-operationen

På baggrund af ovenstående billedet kan det konkluderes, at Read-operationen virker efter hensigten.

De oplysninger, som ønskes i databasen, kan tilføjes ved anvendelse af Create-operationen, og ønsker man at hente oplysninger fra databasen anvendes Read-operationen. En mere dybdegående forklaring af resultaterne kan findes i Dokumentation. De indtastede oplysninger skal også kunne opdateres, til dette har gruppen implementeret en Update-operation. Update-operationen skal finde sted, når en Player spiller videre på sin bruger og optjener eller taber points. Næste afsnit vil komme ind de tests, der udføres for Update-operationen.

11.1.3 Update

Dette afsnit omhandler de resultater, der fremkom under tests af Update-operationen. Update-operationens formål er at opdatere data, som tilhører en Player i databasen. Det er en forudsætning, at en Player skal være oprettet i databasen, før Update-operationen kan køres.

Billedet forneden illustrerer, hvad databasen på nuværende tidspunkt indeholder. Det kan ses på billedet, at de oplysninger, som blev indtastet under afsnittet med Create, stadig er i databasen.

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

Figur 22: Billede af tabel før Update-operationen køres

På baggrund af ovenstående billede kan det konkluderes, at de indtastede oplysninger fra afsnittet med Create, stadig befinder sig i databasen, og Update-operationen kan nu anvendes.

Programmet bliver kørt, og der tjekkes efter i databasens tabel, om oplysningerne er ændret. Billedet forneden illustrerer tabellen efter Update-operationen er kørt.

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

Figur 23: Billede af tabel efter Update-operationen køres

Det kan på baggrund af ovenstående billede konkluderes, at oplysningerne som findes i databasen, ændres til de nye angivne informationer. Det kan konkluderes, at oplysninger er ændret, og at ID'et stemmer overens med det forrige ID. En mere dybdegående forklaring af resultaterne kan findes i Dokumentation.

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.

11.1.4 Delete


Dette afsnit kommer til at handle om de resultater, der fremkommer under test af Delete-operationen. Delete-operationens formål er at slette en Player fra databasen. Det er en forudsætning, at en Player findes i databasen, før en Delete-operation kan køres.

Der bliver tjekket efter, om der er indsat noget data i databasen, dette gøres ved at kigge på tabellerne i databasen. Billedet, som er forneden, viser de oplysninger, som findes i databasen.

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

Figur 24: Billede af tabel før Delete-operationen køres

Det kan konkluderes på baggrund af ovenstående billede, at de opdaterede værdier, som blev indtastet under forrige afsnit med Update-operationen, stadig findes i databasen. En mere dybdegående forklaring af resultaterne kan findes i Dokumentationen. Da det kan konkluderes, kan Delete-operationen blive kørt. Billedet, som kan ses forneden, viser databasens indhold efter, Delete-operationen er blevet kørt.

| | PlayerID | HighScore | UserName | Wallet |
|---|----------|-----------|----------|--------|
|  | NULL | NULL | NULL | NULL |

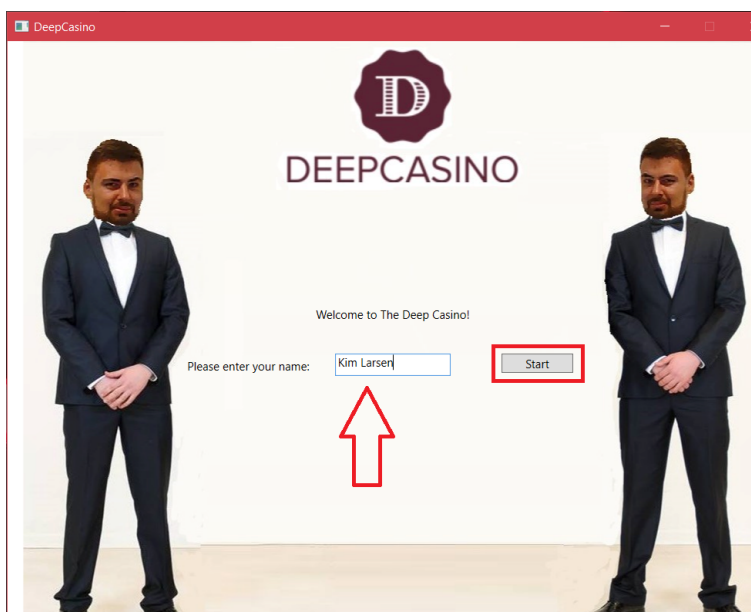
Figur 25: Billede af tabel efter Delete-operationen køres

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.

Det næste afsnit omhandler de resultater, der fremkommer under test af forbindelsesleddet mellem databasen og DeepCasino.

11.1.5 Forbindelsesleddet mellem databasen og DeepCasino - Test

Under dette punkt ses resultatet af forbindelseleddet mellem databasen og DeepCasino. Der er oprette en SQL-forbindelse for at kunne forbinde mellem databasen og DeepCasino. DeepCasino skal kunne sende brugerens data til databasen og for at kunne udskrive det på tabellen i database, skal der benyttet SqlCommand. På Billedet nedenunder ses der en applikations startside, hvor man opretter en bruger. Det TextBox som er markeret med en røde-pil, skal brugeren indtaste sit navn og dermed trykke på start-knappen som er markeret med rødt. Derefter sker det en event i systemet.



Figur 26: Billedet af start-siden af applikationen

På figuren nedenunder ses der en tabel fra databasen, hvor man kan ses brugeren er oprettet. På tabellen ses både brugerens navn(Username) og DeepCoins(Wallet).

| | PlayerID | HighScore | UserName | Wallet |
|---|----------|-----------|------------|--------|
| ▶ | 22 | 0 | Kim Larsen | 1000 |
| ⊗ | NULL | NULL | NULL | NULL |

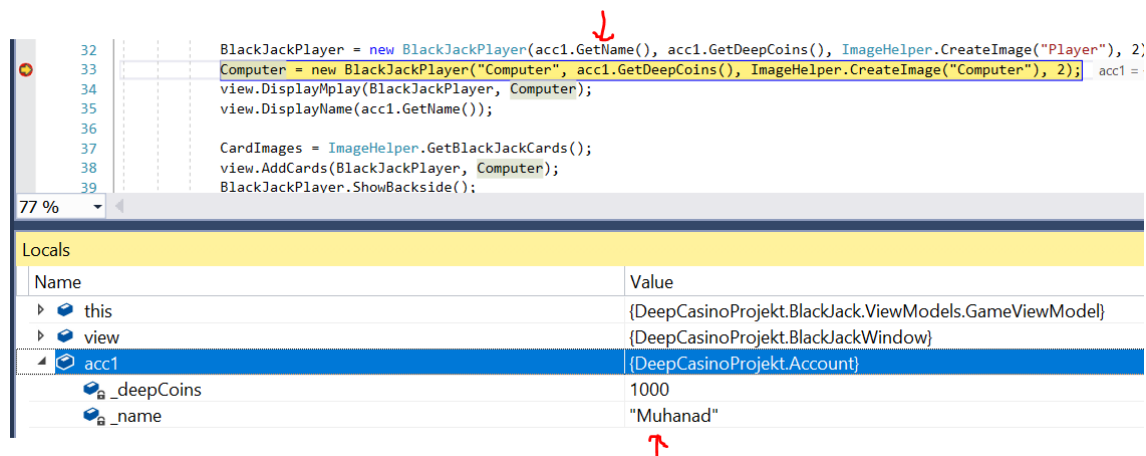
Figur 27: Billede af tabellen fra database

Det kan nu konkluderes, at forbindelsesleddet mellem databasen og DeepCasino fungerer efter hensigten, skal BlackJack-applikations testresultater uddybes. Det næste afsnit omhandler de resultater, der fremkommer under tests over BlackJack.

11.2 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.

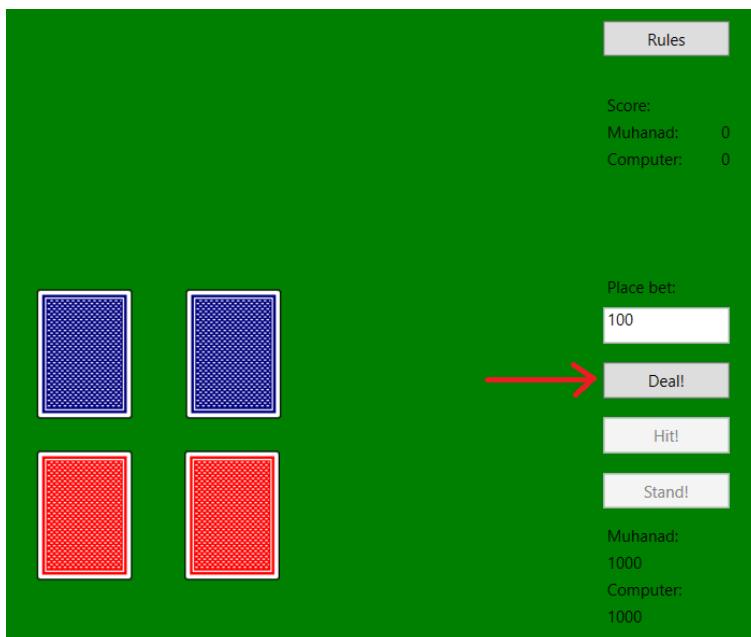
På figur 28 illustreres et eksempel på, hvordan gruppen har brugt debugging for at verificere at navn samt deepcoins bliver tildelt brugeren. De relevante steder at kigge på figuren peges på med to røde pile. Den øverste pil viser den kodelinje, hvori gruppen har sat et breakpoint. Den nederste pil viser det sted man ser, at brugerens navn "Muhanad"og deepcoins er kommet med over i BlackJack-projektet fra opstartsapplikationen. Brugerens Muhanads deepcoins vil også dekrementeres eller inkrementeres alt efter om Muhanad vinder eller taber et spil.



Figur 28: Eksempel på brug af debug.

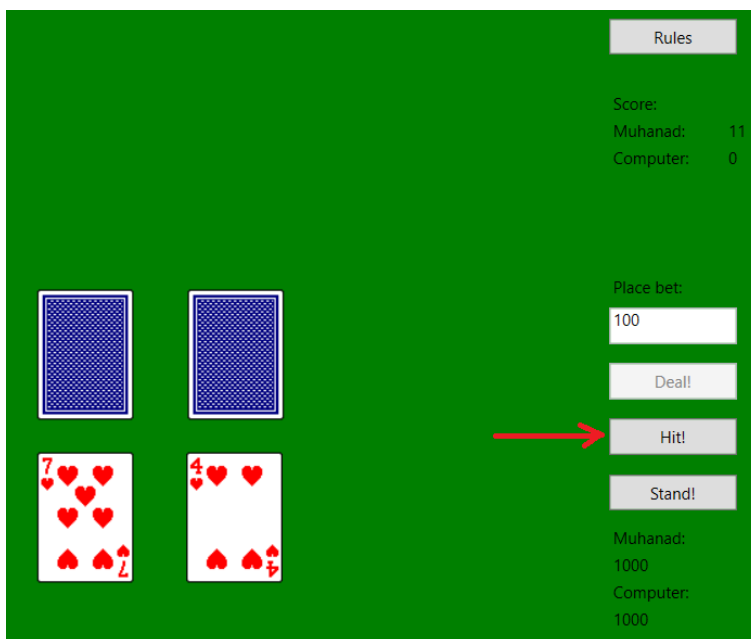
Følgende ønsker gruppen at demonstrere at User Story 3 "Brugeren skal kunne spille BlackJack", fungerer efter hensigten.

På figur 29 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 30.



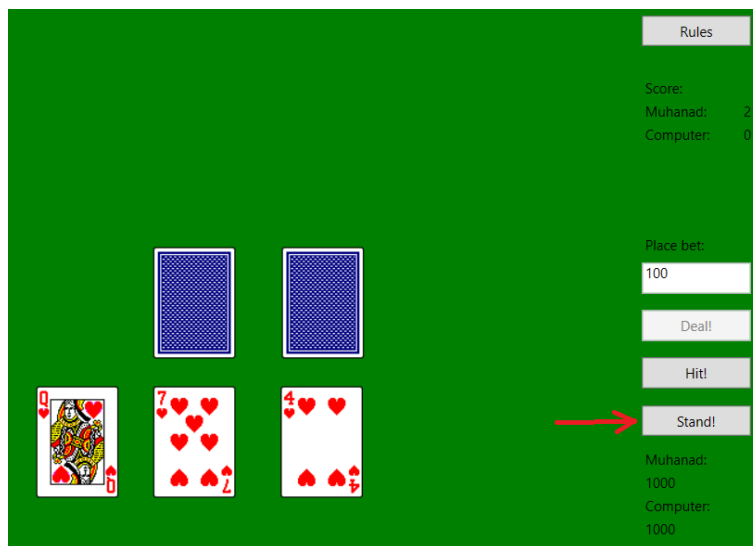
Figur 29: Ved kørsel af BlackJack-applikationen.

Som det ses på figur 30 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 31.



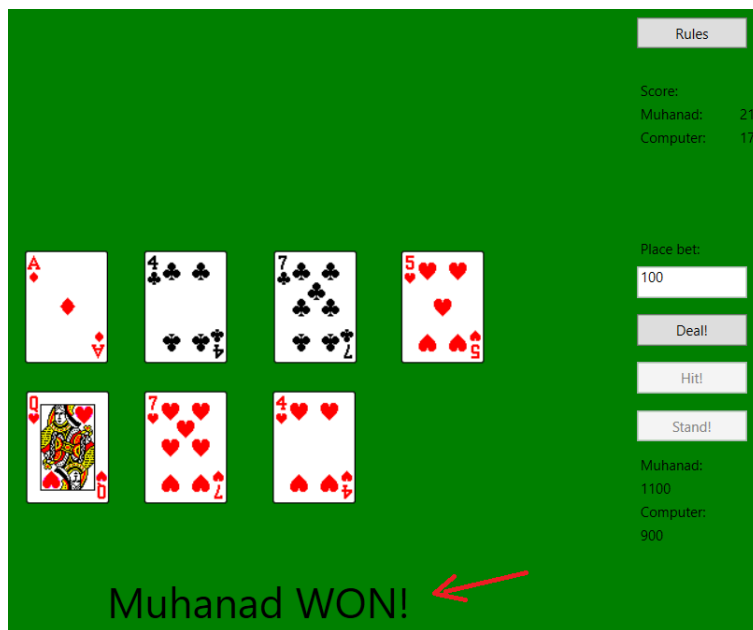
Figur 30: BlackJack-applikationen efter at brugeren trykker "Deal".

På figur 31 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 32.



Figur 31: BlackJack-applikationen efter at brugeren trykker "Hit"

Her (figur 32) 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 32: 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. statisk analyse ses det på figur 33 at gruppen har formået at holde maintainability indekset markant over 20.

| Code Metrics Results | | | | | | |
|--|-----------------------|---------------------|----------------------|----------------|---------------|--|
| Filter: None | Min: | Max: | | | | |
| 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 | |

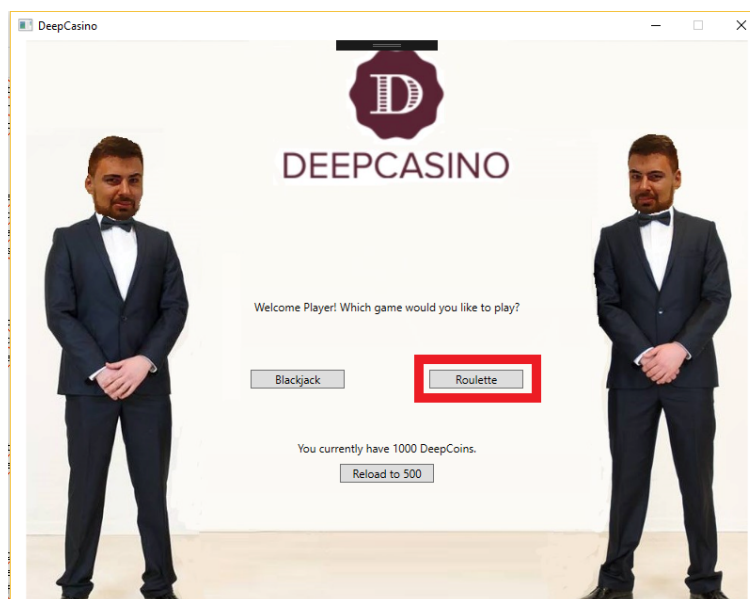
Figur 33: Visual studio statistisk analyse

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.

11.3 Roulette

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

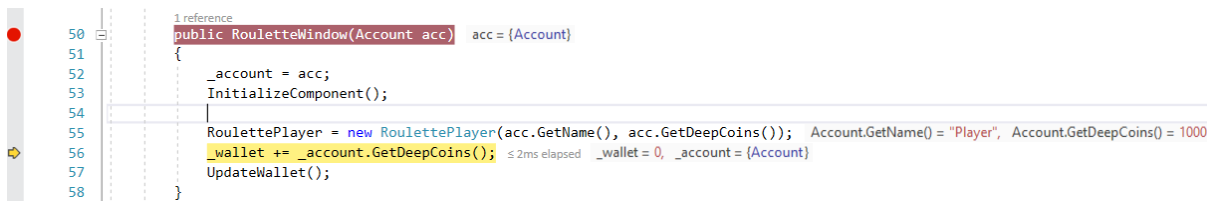
Figur 34 nedenfor, viser opstartsapplikationen. Meget kort illustrerer figuren, hvordan applikationen tilhænger spilleren's konto til Roulette-applikationen. De senere figurer viser gennemgangen af de få steps via debuggeren.



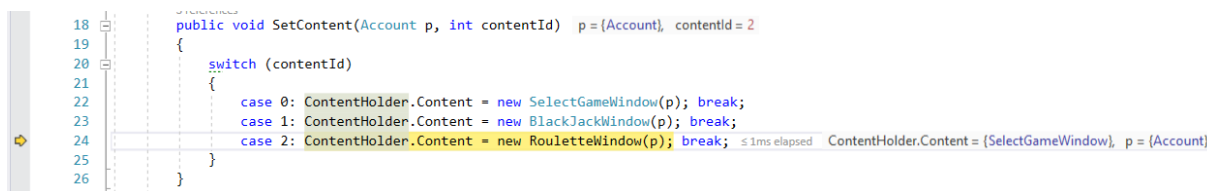
Figur 34: Spiller der vælger Roulette på opstartsapplikationen

Figur 35 og figur 36 nedenfor, er en forlængelse af den øvrige figur. Meget simpelt, viser de **case 2**, som er anvendelsen

af RouletteWindow.



Figur 35: Spiller bliver registreret i Roulette-applikationen



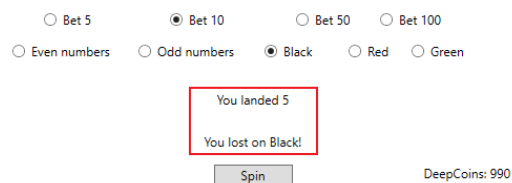
Figur 36: Case 2. Applikationen arbejder med RouletteWindow

Kort efterfulgt af opstartsapplikationen, ses forsiden til Roulette-applikationen. Spilleren får muligheden for at vælge en værdi som spilleren ønsker at satse, samt hvad spilleren ønsker at spille på. Det ses på figur 37 nedenfor, at spilleren vælger **bet 10** og **black**.



Figur 37: Forsiden på Roulette-applikationen - RouletteWindow

Det næste skridt er gennemgangen af spillet roulette. Figur 38 nedenfor vises resultatet, efter spilleren har trykket på Spin knappen. En beskrivende sekvens på dette, kan læses under afsnittet Softwaredesign.

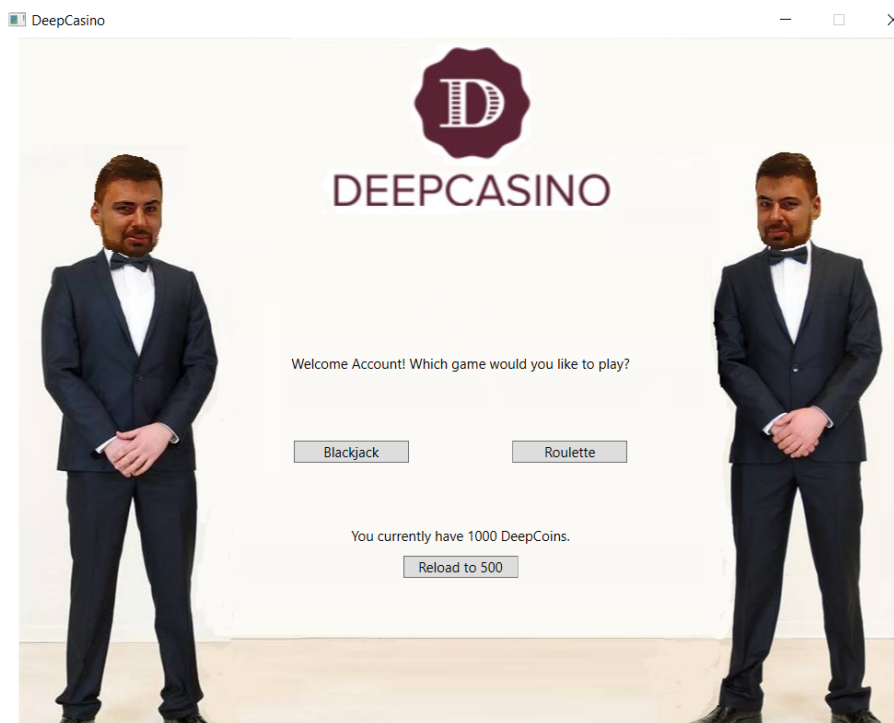


Figur 38: Resultatet på roulette, efter applikationen er kørt

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

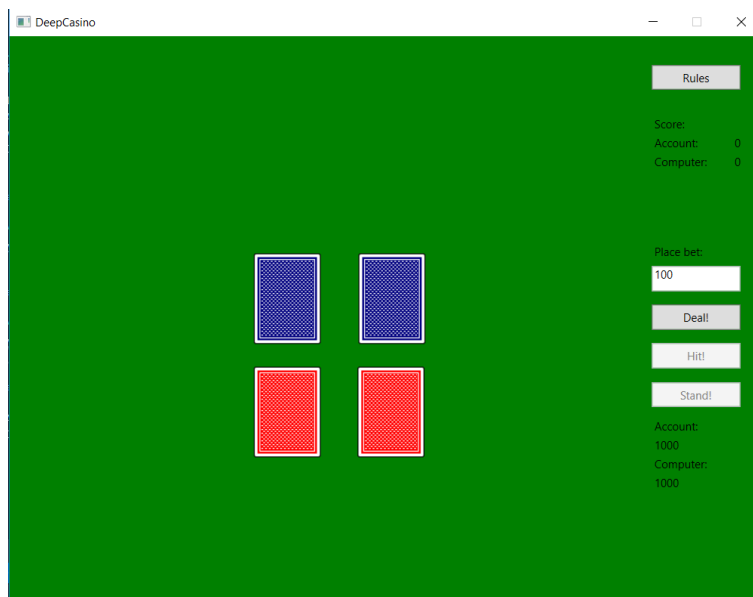
11.4 Opstartsapplikation

Her vises resultaterne for opstartsapplikationen fra de test som blev beskrevet i afsnittet Test. Figur 39 viser applikationens SelectGameWindow, hvori man kan se at knapperne til BlackJack og Roulette er indeholdt.



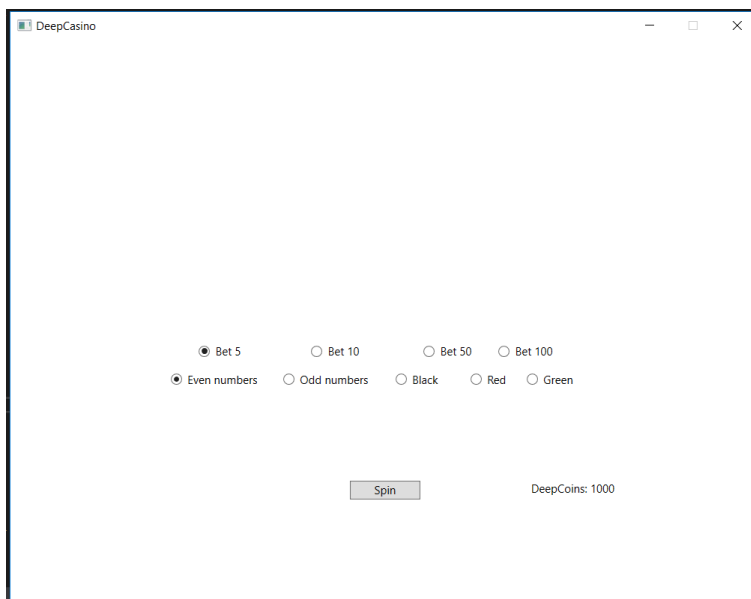
Figur 39: SelectGameWindow

Når gruppen fra SelectGameWindow(figur 39) trykker på Blackjack-knappen åbnes vinduet som det ses på figur 40, hvilket var det forventede resultat.



Figur 40: BlackjackWindow

Ligeledes når gruppen fra SelectGameWindow(figur 39) trykker på Roulette-knappen åbnes vinduet som ses på figur 41, hvilket var det forventede resultat.



Figur 41: RouletteWindow

I forhold til accepttesten er opstartsapplikationen fyldestgørende for User story 5. Udfra de foretagne test vurderes User story 5 som godkendt.

12 Diskussion af resultater

Gruppen har igennem accepttesten opnået en række resultater ved visuelt test og debugging. Der vil i dette afsnit blive diskuteret på gruppens mest essentielle User-Stories og deres betydning i de forskellige dele af projektet. Der vil her ligeledes blive givet en samlet vurdering af de opnåede resultater med relation til projektets problemformulering.

Under udviklingen af databasen er der blevet taget udgangspunkt i user story 2. Det har været en udfordring at implementere databasen i projektet, eftersom der først havde arbejdet med en lokal database. Derfor blev der valgt at anvende en server database, som er udgivet af underviseren i E18I4DAB. Problemet med databasen er, at Update og Delete operationerne ikke virker, grundet tidspres og mangel på erfaring. Create virker næsten korrekt, ment på den måde, at når brugeren indtaster et brugernavn der på forhånd findes på databasen, så vil databasen indsætte en ny række med samme navn, hvilket ikke er hensigten. Dette kunne ikke fikses af samme grund som før.

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 ifm. MVVM i BlackJack-applikationen. 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 i BlackJack-applikationen. 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.

I forbindelse med Roulette havde gruppen planlagt, anvende MVVM for at holde det konsistent med BlackJack applikationen. Dette er dog blevet tilsidesat pga. den manglende tid og ikke-funktionelle kode. Gruppen valgte at prioritere og fokusere på gennemførelse af User story 4 samt anvendelse af FURPS principperne.

Gruppen har diskuteret hvorvidt det tidsmæssigt kunne betale sig, at udføre Unit Tests på de samtlige funktioner, der udarbejdet i Roulette-applikationen. Her besluttede gruppen, at omfanget af disse test ikke vil være tilstrækkelig nok ift. det tidsmæssig pres samt for kompliceret da MVVM design pattern ikke er implementeret.

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. 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.

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.

13 Konklusion

Målet for dette semesterprojekt har været at udføre en casino-applikation, der har til formål at kunne give mulighed for brugeren at anvende virtuelle penge, som kan anvendes i en virtuel shop. Projektgruppen har formået at udvikle et system, som kan indeholde flere gambling-spil, hvilket i gruppens tilfælde er Roulette og BlackJack. Brugeren af applikationen har mulighed for at registrere sig med et brugernavn, samt få tildelt virtuelle penge. Det har ligeledes lykket gruppen at få designet og implementeret en database, hvori brugeren har mulighed for at gemme sit brugernavn. Yderligere har vi gjort det muligt for brugeren selv at vælge hvilket casino-spil han/hun vil benytte sine virtuelle penge på. Dette indebærer, at når brugeren har valgt det ønskede spil, kan han/hun anvende disse virtuelle penge i det pågældene casino-spil. Hermed vil brugeren have mulighed for at forøge eller formindske sit beløb.

Det er op til fremtidigt arbejde at få implementeret databasen således, at de øvrige CRUD operationer virker, såsom Update og Delete. Ligeledes skal der i fremtidigt arbejde implementeres en visuel brugergrænseflade for casino-spillet, Roulette. Slutteligt har gruppen ikke formået at udvikle en virtuel butik, som brugeren kan anvende sine DeepCoins på.

Til udarbejdelse af semesterprojektet har gruppen fulgt udviklingsmetoden, SCRUM og specielt gjort brug af de inddelte sprints. Hver sprint har haft et klart mål og gruppen har sat en tidsramme for at nå dette mål. Dette har resulteret i, at gruppen har formået at færdiggøre en fase inden en ny fase påbegyndes. SCRUM's udviklingsmetode har gjort det ligeledes for gruppen at arbejde iterativt. Gruppen er opdelt i enkelte sprint-teams og har anvendt SCRUM's mødeopsætninger.

Overordnet set har udviklingsprocessen været mærket af et stort forberedelsesarbejde i fællesskab, løbende dokumentation samt hyppige projektmøder til koordinering af projektarbejdet i diminutive grupper. Det har dog været en udfordring for gruppen, at kunne sætte den ønskede tid af til udførelse af de enkelte delmål. Dette skyldes primært pres fra øvrige kurser i dette semester. Alt i alt mener gruppen, at det har været en iterativ læreringsudvikling igennem arbejdet med design og implementering.

14 Fremtidigt arbejde

Gruppen har som nævnt i afsnittet "Afgrænsning" ikke opnået samtlige delmål for projektet. Dette er der flere grunde til, men de to primære årsager ligger i, at vi har siddet fast i de samme problemer for længe, og at gruppen fra starten af dette semesterprojekt har sat alt for høje krav, der kræver mere tid samt planlægning. I dette afsnit vil der derfor komme en forklaring over, hvad der skal udarbejdes færdigt til fremtidigt arbejde. Afsnittende beskrevet nedenstående er delt op så det står i overensstemmelse med de arbejdsfordelinger i gruppen [som ses på side x].

14.1 Database

Til fremtidigt arbejde med databasen, vil der anvendes opdatering af tabellerne. Dvs. hvis brugeren vinder eller taber DeepCoins, så vil der automatisk opdatere brugerens DeepCoins på tabellen i databasen, i stedet for at oprette en ny række med opdateret DeepCoins. Man kan ellers anvende delete metoden til at slette brugeren som er oprettet i tabellen.

I fremtiden kan man lave en liste af brugere på applikationen, hvor man kan se deres HighScore ved at udskrive tabellen fra databasen. Den relationelle database kunne laves om til en NoSQL dokument database, så man kunne benytte os af Swashbuckle og Azure Cosmos. At det er en dokument database vil det betyde, at der kan tillades at arbejde på en web API.

14.2 BlackJack

En udbygning af gruppens BlackJack applikation ville være at implementere alle de officielle funktionalitet. De manglende funktionaliteter er split og double, der ikke er blevet udarbejdet grundet tidspres. Split indebære en teknisk kompliceret kode, og er derfor blevet afgrænset væk, men gruppen har dog gjort sig overvejelser om, hvordan den skal implementeres i fremtidigt arbejde.

Void splitDeck(Player)

- **Parametre:** Brugerens kort som skal splittes.
- **Returværdi:** Ingen
- **Beskrivelse:** Hvis brugeren har fået uddelt to identiske kort med samme værdi af dealer sp har brugeren mulighed for at få splittet dem op til to separate hænder. Brugerens får herefter mulighed for at spørge dealer om et kort mere til hver af disse hænder. Yderligere skal brugerens indtastet DeepCoins beløb fordobles.

14.2.1 MVVM

Til fremtidigt arbejde med MVVM vil det være oplagt at anvende RelayCommand, der har til formål at implementere ICommand. Dette vil specielt gavne koden ift. udvidelse af flere kommandoer da gruppen vil få mere overskuelige og mindre duplikeret kode i applikationen. Grundet gruppen ikke anvende mere end tre kommandoer har vi ikke set de

nødvendigt at sætte tid af til at implementerer relayCommand. En yderligere finpudsning af anvendelse af MVVM design pattern vil have været at fjerne alt kode med undtagelse af constructoren i BlackjackView.cs filen.

14.2.2 Tests

Gruppen har primært haft fokus på anvendelse af statisk analyse og debugging i forbindelse med test af kodens funktionalitet. Der vil dog bestemt blive kørt unit test til fremtidigt arbejde, hvor gruppen vil have mulighed for at dokumentere at de udarbejdede metoder returnerer det ønskede resultat. Yderligere vil der til fremtidigt arbejde blive anvendt C.I for hele gruppens applikation så der kan iagttages hvor meget coverage gruppen har fået dækket af applikationen.

14.3 Roulette

I forbindelse med udviklingen af roulette-applikationen, var det hensigten at der skulle laves en implementering af MVVM design pattern, dels for at gøre koden mere overskuelig, ved at dele den op i forskellige lag, men også for at give andre udviklere mulighed for at arbejde på tværs af applikationen, i tilfælde af udvidelse, uden at disse ville konflikte med hinanden.

En anden tanke der kunne gøres, er det visuelle i applikationen. Eksempelvis kunne der tilføjes et grafisk design af et hjul, der drejer når Spin-knappen bliver trykket på. I stedet for at udskrive resultatet lige efter eksekveringen af spinButton_Click() funktionen, ville den udskrive det efter et hvis antal tid, når hjulet er stoppet med at dreje. Blandt andet kunne man tilføje lydeffekter til det visuelle.

Yderligere kunne man tilføje de resterende spillemuligheder, som blev afgrænset under Analyse afsnittet. Derudover også en ekstra knap, der kan åbne op for et nyt vindue, som vil vise spille-reglerne for The Deep Casino's roulette.

14.4 Opstartsapplikation

I forhold til projektets mål ville det klart være at udvikle funktionaliteten The Deep Shop til opstartsapplikationen, da gruppen har to user stories der direkte afhænger af den. Ideen til hvorledes dette skal opnås, er ved at udbygge applikationen til at have endnu et vindue, opbygget tilsvarende SelectGameWindow. The Deep Shop skal kunne tilgås når brugeren er logget ind, derfor ville det være oplagt at have en knap på SelectGameWindow tilhørende shoppen. Metoden til knappen kunne se således ud:

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

Listing 18: Metoden DeepShop_Click()

Som man kan se, er algoritmen ens til sammenligning med eksempelvis BlackjackButton_Click, som er beskrevet i implementeringen. Det mest essentielle er at contentId i SetContent sættes til "3", da gruppen allerede bruger 0, 1

og 2. Konceptet The Deep Casino lægger op til at dette vindue skal indeholde præmier, som brugeren kan bytte sine deepcoins til.

15 Ordliste

| Ord | Betydning |
|--------------------|---|
| BlackJack | At hit-, stand- og deal-funktionen implementeret. |
| Business logic | I business logic bliver alt dataen behandlet. |
| C.I-server | Continuous Integration er en softwareudviklingspraksis, hvor medlemmer af et hold integrerer deres arbejde ofte på en server. |
| Code coverage | Code coverage er en måling af hvor mange linjer af gruppens kode udføres, mens de automatiske tests kører. |
| DeepCoins | Virtuel valuta tilknyttet The Deep Casino. |
| DeepShop | Virtuel Butik tilknyttet The Deep Casino. |
| SRP | Single Responsibility Principle. |
| OCP | Open-Closed Principle. |
| LSP | Liskov's Substitution Principle. |
| ISP | Interface Segregation Principle. |
| DIP | Dependency Inversion Principle. |
| GUI | Kurset GUI (Graphical User Interface) programmering. |
| Cohesion | Beskriver hvor connected eller relateret funktioner af modulerne er. |
| Presentation logic | Dette beskriver den bagvedliggende kode for view. |
| ReSharper | Et plugin til Visual Studio. |
| Dynamic Analysis | Giver feedback til programmøren, når programmet kører, fx. i hvor lang tid. |
| Roulette | Velfungerende applikationen der kan spille på de forskellige muligheder. |
| 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. |
| CRUD operationer | De 4 basiske operationer, Create, Read, Update og Delete for en database. |

16 Referenceliste

16.1 Database referenceliste:

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

16.2 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>

16.3 Roulette referenceliste:

16.4 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