Dungeons and Gnoblins

Semesterprojekt

Aarhus Institut for Elektro- og Computerteknologi

Authors:

Magnus Blaabjerg Møller, 202006492
Sune Andreas Dyrbye, 201205948
Morten Høgsberg, 201704542
Rasmus Engelund, 202007668
Anders Hundahl, 202007859
Oscar Dennis, 202009975
Jacob Hoberg, 201807602
Luyen Vu, 202007393

Vejleder: Jung Min Kim (Jenny)

Anslag:

Afleveringsdato: 03-06-2022 Eksamineringsdato: 22-06-2022

Resumé

I 1980 blev det text-based adventure game Zork, udgivet for første gang. Spillet tillod udviklere at fortælle en kompliceret historie, påtrods af manglende 2D og 3D grafiske værktøjer. Ved at benytte de samme koncepter, som Zork brugte i 1980. Med brug af moderne udviklings metoder og værktøjer som agile, scrum, continuous integration og moderne backend og databaser har projektet udviklet sin egen udgave af en gammel indie klassiker.

Gennem projektet blev det klar fra starten, at tidlig integration mellem projektets fire grene, frontend game engine, backend og database, var kritisk for projektets succes. Dette skyldes at processen når mange små ændringer integreres i projektet er simplere end proceseen der skal til at integrere store ændringer. Selvom projektet ikke har brugt fult automatiseret continuous integration har ofte testning af det integrerede system sikret at afvigelser mellem individuelle moduler ikke voksede til en størrelse som gjorde det uoverskueligt at løse. Det hyppige integration har også hurtigt givet os et grundlæggende system, som alle efterfølgende moduler kunne bygges ovenpå. Dette medførte at alle efterfølgende moduler, som nødvendigvis virkede med basen (som de var bygget på) integrerede nemt med hverandre.

Projektet følger den originale arkitektur ganske tæt, med kun få ændringer. Disse ændringer er der taget højde for senere i designet af projektets forskellige komponenter.

Slut produktet er således en gennemtestet prototype til et spil, der med tiden kunne være blevet et fuld udbygget text-based adventure game, og det endte med at blive et gennemført proof of concept.

Abstract

1980 saw the premier release of the text-based adventure game, Zork, which allowed the developers to tell a complex story despite the lack of advanced 2D and 3D graphics. Utilising the same concepts as Zork in the 1980 with the addition of modern development methods and tools like agile, scrum, a subset of continuous integration and a backend database solution, the projekt has successfully created its own take on an otherwise old indie game genre.

Throughout the projekt it became clear from early iterations that early integration between the projects four main branches, frontend, game engine, backend and database would be critical for success as it simplified the final integration, when doing many small interations instead of a few large once. While the projet did not use an automated continuous integration, frequent integration testing ensured that no mismatch between modeules grew into an unsurmountable obstacle. The integration also allowed for an early base which could be expanded on. This in turn ensured that all susequent systems (which by neccessity conformed to the already established base) integrated well with eachother.

The project mostly follows the original architecture with only few deviations, that are otherwise accounted for in the design and the projects many components.

The end product is a well tested, prototype of what could later become a full fledged text-based adventure game, and it turned out to be an excellent proof of concept.

	Morten	Sune	Anders	Jacob	Oscar	Luyen	Rasmus	Magnus
Kravspecifikation	x	x	x	x	x	x	x	x
Frontend Arkitektur		x	x					
Game Engine Arkitektur	x					x		
Backend Arkitektur				x				x
Database Arkitektur					x		x	
Frontend Design		x	x					
Game Engine Design	x					x		
Backend Design				x				x
Database Design					x		x	
Frontend Implementation		x	x					
Game Engine Implementation	x					x		
Backend Implementation				x			x	x
Database Implementation					x		x	
Frontend Modul Test		x	x					
Game Engine Modul Test	x					x		
Databse Modul Test					x		x	
Integrationstest	x	x	x	x	x	x	x	x
Accepttestspecifikation	x	x	x	x	x	x	x	x

Contents

1 Introduction

1.1 Indledning

Populariteten for videospil er på et højdepunkt som aldrig før [**REFERENCE**], og denne tendens ser kun ud til at fortsætte. Dette har medført, at producenterne er begyndt at lave remakes af deres gamle klassikere, hvor de så tilføjer nogle moderne aspekter til spillets funktionalitet [**REFERENCE**]. Det er her motivationen for projektet kommer ind:

Dette projekt vil tage udgangspunkt i den klassiske genre kaldet Text-based adventure game. Denne genre blev for alvor kendt tilbage i 1980'erne, da spil serien "Zork" blev udgivet. Zork var en enorm succes[**REFERENCE**]., og skubbede for alvor denne genre i gang. Dette gav anledning til at lave en efterfølger på det populære spil, som udkom i det sene 80'ere. Her var spillet blevet udvidet med flere features. Fra moderene forbrugeres synspunkt har Zork spillene nogle problemer. Blandt andet den manglende evne til at lave "save games" som kan gemmes i cloud. Dette ville i dag gøre det muligt at kunne tilgå et specifik spil fra en anden enhed, uden at man skal flytte filer. Der var desuden ingen authentication, hvilket betyder at enhver der havde adgang til den pågældende PC, ville være i stand til at fortsætte eller slette gemte spil. Dette projekt vil forsøge at genopfriske den text-based adventure genre, ved at lave et nyt spil. Der vil blive taget inspiration fra Zork serien, ved at grundbasen for spillet, vil forblive nogenlunde det samme. Dog har projektet til hensigt at forsøge at tilføje nogle moderne aspekter til spillet - Heriblandt en opdaterende GUI der viser et map, en backend server til at save games og for authentication.

1.2 Problemformulering

Text based RPG er som sagt en spilgenre som var yderst populær i 80'erne. Målet med dette projekt er at genoplive spil genren Text based RPG, ved at udvikle et spil som kombinerer den klassiske spilgenre med ny teknologi. Problemet med tidligere Text based RPG spil, er at der har været begrænset mobilitet i forhold til at gemme og hente spillet på forskellige enheder. I forbindelse med denne problemstilling kan det være svært for brugeren at skabe sig et overblik over sine fremskridt i spillet efter en potentiel pause herfra. Hvad angår sikkerhed er det heller ikke noget der traditionelt har været fokus på i forbindelse med genren. På baggrund af disse udfordringer ved Text based RPG spil, vil dette projekt undersøge følgende problemstilling:

Er det muligt at genoplive den klassiske spilgenre Text-based RPG, igennem tilføjelse af moderne teknologier såsom cloudsaving, således at genren bliver attraktiv for den nye generation af spillere samtidigt med at den klassiske spiloplevelse fastholdes?

https://www.protocol.com/video-game-remaster-remake-boom

2 Systembeskrivelse

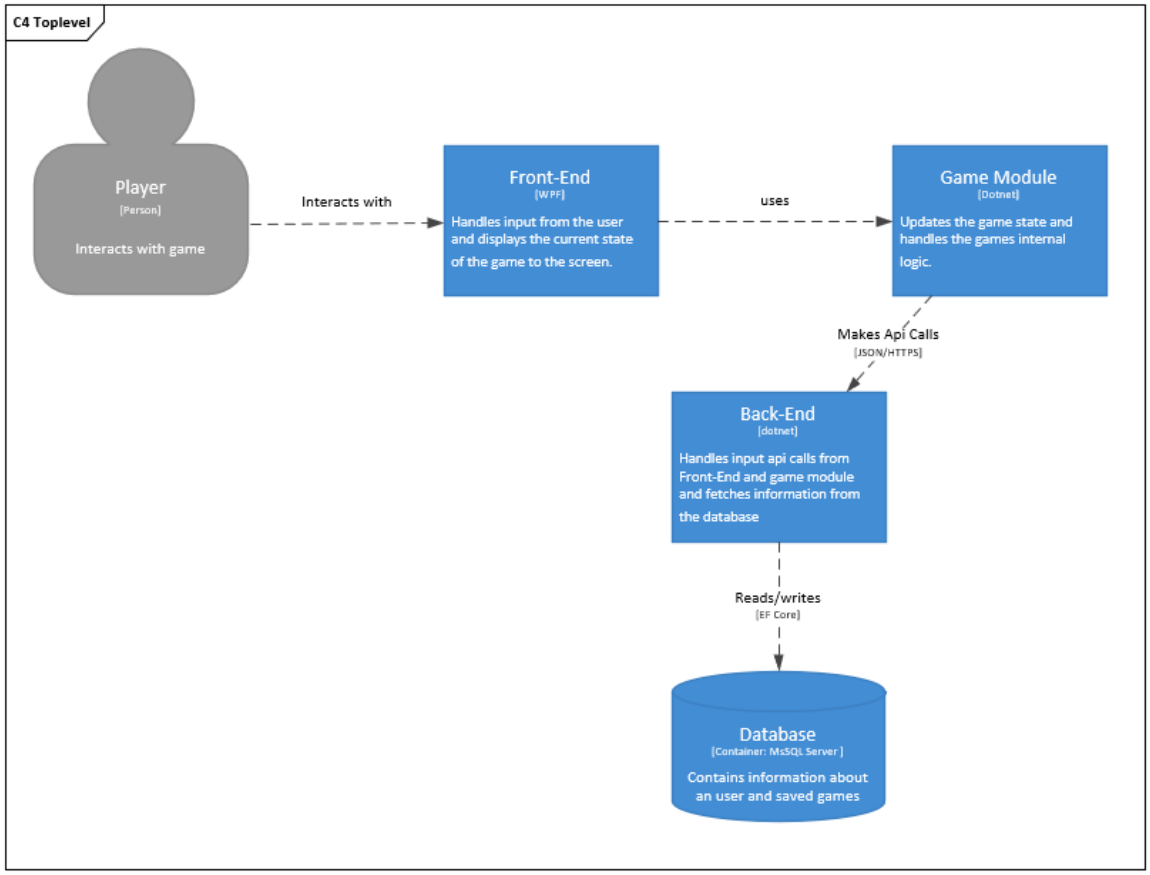

Figure 1: C4 top level diagram, som viser kommunikation mellem systemets segmenter

På Figure 1 ses systemets toplevel arkitektur. Denne består af en bruger som interagerer med systemet gennem frontendapplikationen som skrives i WPF. States i denne applikation styres af Game module som holder styr på hvor spilleren befinder sig, hvilke items der er samlet op og andre nyttige information som skal bruges gennem spillet. Spillets backend benyttes primært til bruger-authentication og som bindeled til databasen. Databasen indeholder oplysninger om blandt andet brugere, de gemte spil og oplysninger om historien for de forskellige rum.

3 Kravspecifikation

I det følgende afsnit beskrives kravene for systemet. Afsnittet indeholder et udpluk af de funktionelle og ikke funtionelle krav, som omhandler det samlede system. Til slut i afsnittet afgrænses projektets omfang her præsenteres en MoSCoW-analyse prioritering over kravene, samt hvilke rammer som sættes for projektets omfang. For en fuld kravspecifikation henvises til bilag **(Mangler Ref til bilag)**.

3.1 Funktionelle krav - User Stories

Her beskrives systemets funktionelle krav. Kravene beskrives i form af User Stories. Der er her taget udgangspunkt i User Story 1,2, 15 og 18, da disse indebærer alle systemets dele samt den mest centrale funktionalitet.

User Story 1: Log in
Som Bruger
Kan jeg logge ind
For at kunne se en Main Menu så jeg kan spille spillet.

User Story 2: Opret profil
Som Bruger
Kan jeg oprette en ny profil
For at jeg kan logge ind på spillet.

UserStory 15 : Save Menu - Save Game
Som bruger
Kan jeg trykke på et eksisterne spil, ændre navnet og trykke "Save Game"
For at gemme spillet.

UserStory 18 : Main Menu - Load Game - Load
Som bruger
Kan jeg trykke på "load game"
For at komme ind i spillet og fortsætte med at spille det valgte spil

3.2 Ikke-funktionelle krav

Systemets ikke-funktionelle krav opstilles efter en række kategorier med inspiration fra modellen FURPS+. Igen tages her udgangspunkt i et udpluk af de ikke-funktionelle krav, som indeholder de mest kritiske punkter.

DATABASE:

- Skal kunne gemme maksimalt 5 save games
- Skal kunne loade et spil indenfor maksimalt 5s
- Skal gemme hvilke genstande man bruger lige nu
- Skal gemme hvor meget liv man har tilbage.
- Skal gemme hvilke fjender man har slået ihjel.

- Skal gemme hvilke puzzles man har løst.
- Skal gemme hvilke rum man har været i.

GAMEPLAY:

- Spillets kort skal holde styr på hvilke rum man kan komme til for et givet rum.
- Spillets kort skal kun vise de rum som spilleren har været i.
- Spillets kort skal, hvis spilleren har været i alle rum vise alle rum.
- Et rum kan have maksimalt 4 forbindelser til andre rum.
- Et rum skal have mindst 1 forbindelse til andre rum.
- Alle Rum skal kunne nås fra ethvert andet rum, måske ikke direkte, men man skal kunne komme dertil.
- Spillerens rygsæk skal kunne indeholde alle spillets genstande.
- Spilleren skal have mulighed for at bruge ét våben og én rustning af gangen.
- Spilleren skal have mulighed for at skifte hvilket våben og hvilken rustning der bruges.

COMBAT:

- Når spilleren/fjenden prøver at slå, rammer man kun hvis man på sit angreb slår højere end modstanderens rustningsværdi. Dette afgøres af et simuleret 20-sidet terninge kast, hvortil der lægges en værdi til, korresponderende til spilleren/fjendens våben bonusser. **FUNKTIONEL?**
- Hvis spilleren/fjenden rammer, bliver skaden bestemt af et/flere simulerede terningekast, afhængigt af hvilket våben der bruges
- Hvis spilleren, når nul liv inden fjenden, så dør spilleren og spillet er tabt.
- Hvis fjenden, når nul liv inden spilleren, så dør fjenden og spilleren kan nu frit udforske rummet, som fjenden var i.
- Hvis spilleren drikker en livseleksir bliver spillerens nuværende liv sat til fuldt.

PERFORMANCE:

- Spillet skal respondere indenfor maksimalt 5s
- Spillet må ikke have mere end én kommando i aktionskøen af gangen

3.3 Afgrænsning

I dette projekt afgrænses systemet på følgende måder. Der vil ikke være fokus på at hoste systemets backend (Web API og Database) på en ekstern server. Dette er valgt for at holde fokus på selve udviklingen af systemet. Dette er noget som tages videre i det fremtidige arbejde.

3.3.1 MOSCOW krav

I dette afsnit er opstillet en MoSCW analyse af de krav som er opsat til projektet.

MUST

- have en GUI.
- have en Database.
- Have Netværkskommunikation.
- Have en serie af sammenhængende rum.
- Have et start og slut rum som ikke er det samme rum.
- Spillet skal kunne gemmes på en Database.
- Skal kunne hente save game fra databasen.
- Skal have en authentication system (Accounts).
- Hvert Rum skal bestå af et beskrivende element og en serie af actioner.
- Spillet skal have instillinger.
- Spillet skal have en Character.
- Spillet skal kunne tage imod user input.
- Spillet skal være udviklet til Windows.
- Spillet skal være testbart i.e. skal være designet med testing in mind.

SHOULD

- Have Fjender
- Have Items
- Have Kamp systemer
- Have End Screen
- Have Character Stats

COULD

- Have Level development
- Have Procedual verden
- Have Text Parser
- Have Sværhedsgrad
- Have sikkerhed

WON'T

- Have Grafik

4 Metode og Proces

I nedstående afsnit fremlægges metoder og processer brugt til udviklingen af Dungeons and Gnoblins spillet kort, for at se mere om metode og proces henvises der til den vedhæftede process rapport **INDSÆT REFERENCE HER**.

4.0.1 Projektforløb og møder

Projektets endelige mål har været implementeringen af et text-based adventure game. Hertil har gruppen gjort brug af agile develepment processen til at imødekomme dette mål.
Først har gruppen specifiseret Kravspecifikationerne for spillet, med ønsker om hvilke features spillet har skulle inkludere.
I løbet af projektforløbet blev der afholdt to faste ugentlige møder. Et internt gruppemøde og et vejledemøde. Til hvert møde blev der gennemgået hvad der var blev arbejdet med, hvilke udfordringer der forekom samt hvad der fremadrettet skulle arbejdes med for hver af gruppens medlemmer. Dette gjorde vi i form af SCRUM for at give et bedre overblik for projektets fremgang og dermed vurderer om gruppen var bagud eller foran ift. Gruppens tidsplan.

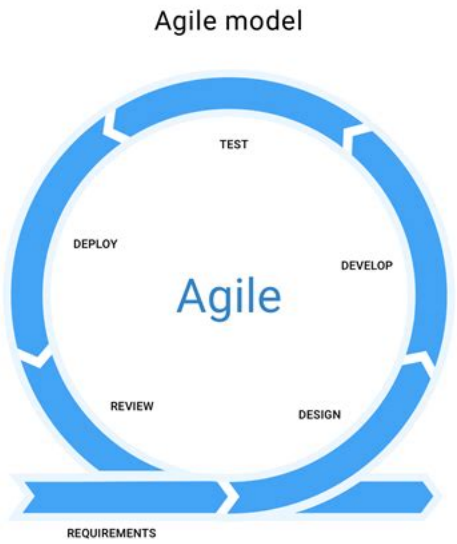

Figure 2: Viser et billed af agile modellen som består af mange iterationer af design, develop, test, og deploy. Dette minder om continuous integration og har hindret mange problemmer for udviklingsforløbet

SCRUM og AGILE bringer klarhed til medlemmerne om deres roller og opgaver over en kommende tidperiode med en backlog over opgaver, som skal færdiggøres over et sprint (1 uge). Ugentlige opdateringer og møder omkring potentielle problemmer udviklingsforløbet betyder at gruppen har kunne tage hånd om evt. problemmer tidligt i forløbet og derved løse dem før de har udviklet sig til større problemmer. Til styring og implementering af souce code er der i gruppen blevet benyttet git som et versions styrings system.

4.1 Modellering

Projektet benytter UML til at beskrive og modellere software arkitektur og design, hvilket gør det nemt at simplificere og visualisere strukturen på software løsningen.
Selve arkitekturen er vist med C4 modellen som giver et lageret indblik i både akitekturen på et højt niveau, men med evnen til at give en detaljeret beskrivelse af systemmets komponenter og deres kommunikation med hinanden.

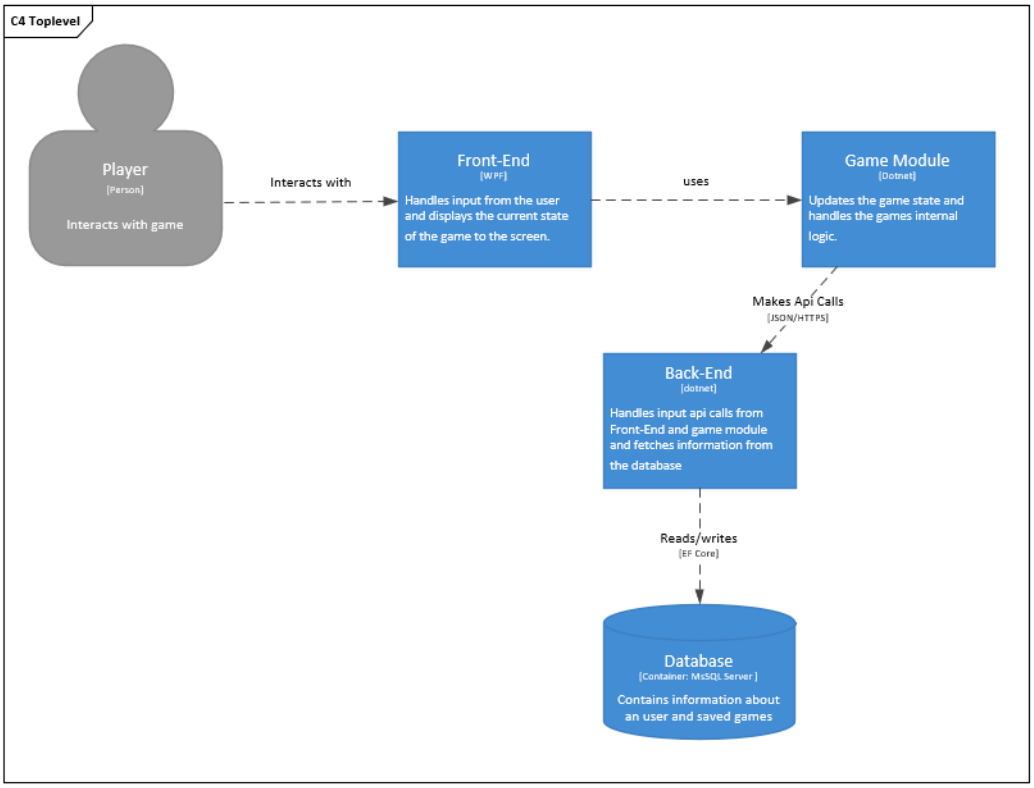

Figure 3: Et eksempel på en C4 model, og det overblik et sådan diagram kan give over et system.

STM og SD diagrammer er brugt til at give programmets udførsel struktur, og fungerer, som en hjælp til at visualisere programmets forskellige states og flow of execution. Det giver et nemt overblik over hvordan funktionskald mellem forskellige komponenter har skal fungere således, at der ikke opstår misforståelse grupperne imellem.
Generelt er arkitekturen opbygget som "pseudo" diagrammer. Dvs. de ligner ikke fuldstændigt det endelige design, men har til formål at vise den overordnede tankegang i systemet og samtidig virke som et udviklingsværktøj til at videreudvikle systemet. Da der i gruppen er blevet arbejdet iterativt, er der forskelle mellem diagrammer for arkitektur og design, da der er blevet tilføjet flere moduler og funktioner undervejs. Den overordnede tanke i projektet er der dog ikke blevet ændret på.

4.1.1 Iterativ Udviklingsforløb

AGILE og continuous integration fungerer på en naturlig iterativ måde, der tillader ændringer til måde designet og implementeringen undervejs i udviklingsforløbet. Under hvert SCRUM møde er der taget stilling til om, der skulle laves ændringer i gruppens tilgang til projektet, altså om en implementering skulle ændres.

5 Analyse

Der vil i følgende afsnit blive præsenteret et kort udsnit af nogle af de analyser, der er blevet foretaget i projektet. Analyserne har til formål at danne et fornuftigt grundlag for videre udvikling af projektet med mindskede risici. Der er foretaget teknologiundersøgelser, for at der kan gives det bedste teknologiske og metodemæssige bud på de respektive problemer der skal løses.

5.1 Teknologi Undersøgelser

Der er i teknologiundersøgelserne blevet undersøgt forskellige løsninger til diverse valg der skulle træffes tidligt i udviklingsfasen.
For det første skulle der træffes et valg om hvilket framework Frontend'en skulle udvikles i. Valget blev hurtigt indsnævret til to kandidater, nemlig .Net og Unity. Kort sagt blev der fundet frem til at med den opdeling af arbejdet, som gruppen ønskede at bruge, gav det mest mening at bruge et framework med en skarp opdeling af Front-end og Back-end. Dette kunne .Net opfylde. Derudover blev der i undervisningen på 4.semester undervist i .NET frameworket, således at der kunne fokuseres på at lave projektet og ikke lære frameworket at kende.

5.2 Teknologiundersøgelse: Database SQL/NOSQL

I dette afsnit diskuteres mulighederne for oprettelsen af en database. Heri vil der diskuteres fordele og ulemper ved anvendelsen af henholdsvis SQL eller No-SQL.

5.2.1 SQL

Den første mulighed for systemets database er at anvende en SQL baseret database. Denne form for database er relational, da den har en samling af data med forudbestemte forhold derimellem. Genstande organiseres I tabeller som indeholder information om objektet. Yderligere er SQL kendt og vel dokumenteret, som resulterer i at denne type database har et stort fællesskab bag den, med mulighed for online støtte til arbejdet med den. Ydermere fungerer SQL også med ACID-principperne.

5.2.2 No-SQL

Et andet alternativ ville være en database baseret på No-SQL. Denne har, modsat SQL, fleksible datamodeller, som gør det nemmere at lave ændringer i databasen efterhånden som kravene til databasen ændres. Yderligere har No-SQL også horizontal scaling, som betyder at hvis der rammes den nuværende servers kapacitet, så har man muligheden for at tilføje mindre ekstra servers, hvor at man ville skulle migrere til en større server i SQL. Hernæst er NoSQL data ofte opsat efter queries. Sådan at data der skal hentes sammen, ofte er opbevaret sammen. Dette resulterer i hurtigere queries.

5.2.3 Konklusion for Database analyse

I dette foranstående segment blev der diskuteret fordele og ulemper ved anvendelse af SQL kontra No-SQL. Hertil bliver der opstillet et valg om hvilket af disse to, der ville være bedst egnet til dette system. Valget vil blive taget senere i design fasen for databasen til projektet. [**Yalantis**] [**IBM**] [**MongoDB.com**]

Yderligere forklaring vedr. de teknologiske undersøgelser kan findes i teknisk bilag **MANGLER REF HER**

6 Arkitektur

6.1 Systemarkitektur

Dette afsnit forklarer hvordan systemets toplevel arkitektur er opbygget. Denne består af en bruger som interagerer med systemet gennem frontendapplikationen som skrives i WPF. States i denne applikation styres af Game modul som holder styr på hvor spilleren befinder sig, hvilke items der er samlet op og andre nyttige information som skal bruges gennem spillet. Spillets backend benyttes primært til bruger authentication og som bindeled til databasen. Databasen indeholder oplysninger om blandt andet brugere, de gemte spil og oplysninger om historien for de forskellige rum. For at se en visuel repræsentation af hvordan de forskellige moduler snakker sammen se Figure 10

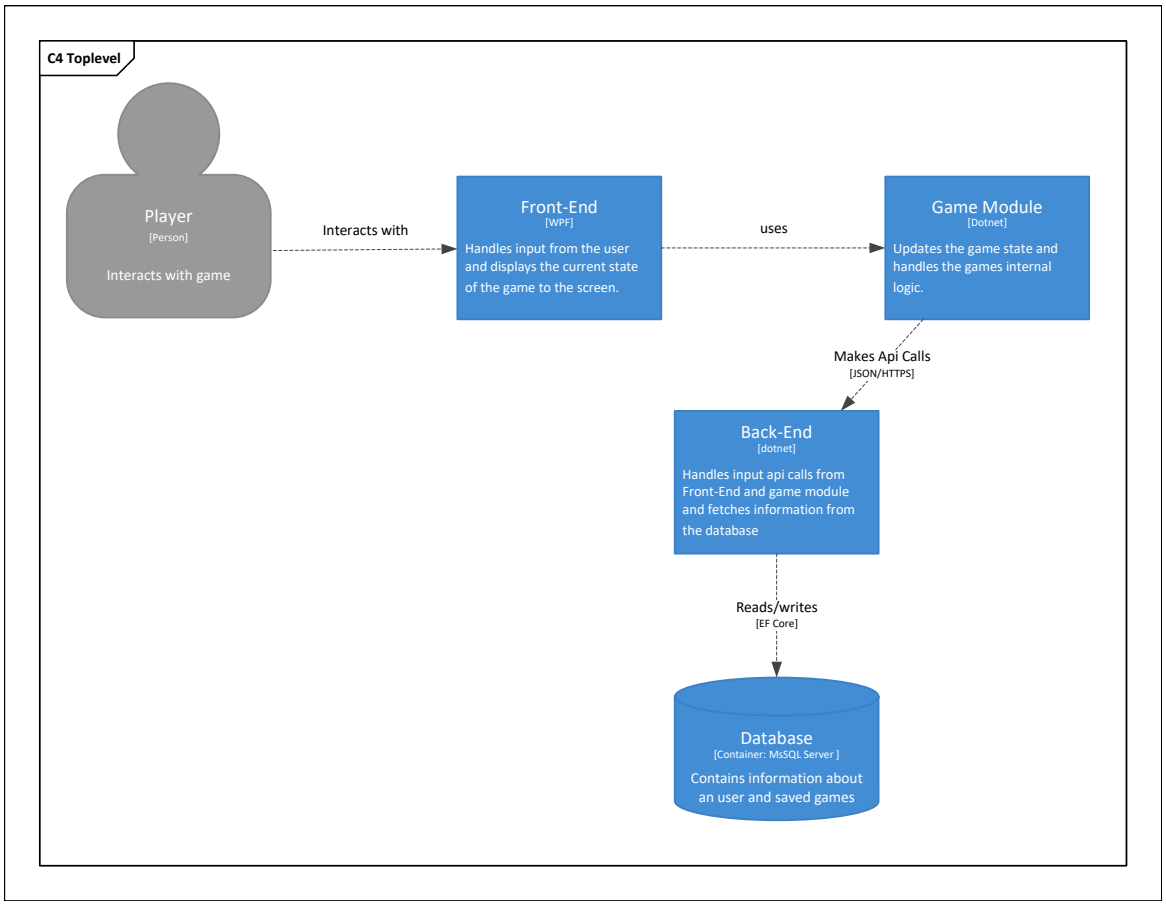

Figure 4: C4 Top-Level diagram for systemets arkitektur. Her ses et diagram for systemets Top-Level arkitektur, hvori der er skabt et overblik over hvilke moduler der er til stede i systemet og hvordan de kommunikere. Heri er der også tilføjet kommunikationsmetode for de forskellige forbindelser.

6.2 Front-end Arkitektur

Front-end applikationen er det man kan kalde, brugerens vindue til spillet, det er i dette modul at brugeren vil få alt sit information og vil få mulighed for at lave inputs til spillet. Det er yderligere her at der skal tages hånd om brugerens input således at de rigtige funktioner i Game Enginen bliver kaldt, når brugeren trykker på en vilkårlig knap.
Front-end'en er opbygget af et hav af forskellige skærme og menuer, og for at give et overblik over

disse er der lavet følgende C3 model for Front-end'en (Figure 5), som giver en idé om hvilke skærme og menuer der kan gå til hvilke andre menuer/skærme. Udover dette fortæller modellen også om hvordan og hvilke skærme og menuer der snakker med noget uden for Front-end'en selv f.eks. skal der ved Login/Register kontaktes databasen for at få verificeret logind-oplysninger, og samtidig skal der ved save og load-game hentes en liste af gemte spil i databasen hvorefter der skal henholdsvis skrives og hentes fra databasen alt efter om man gemmer eller henter et spil.

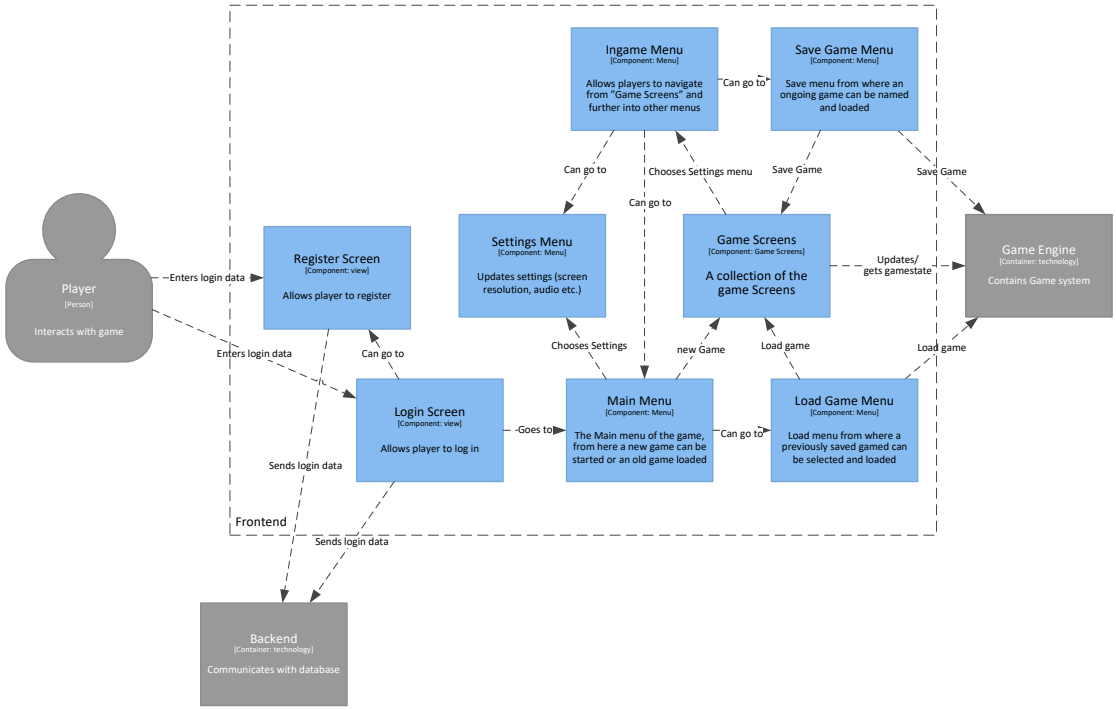

Figure 5: C3-Model for Front-end. Modellen fortæller hvordan man kan navigere igennem forskellige menuer og hvilke menuer der kan føre til hvad. Derudover kan man se hvilke blokke der snakker ud af front-end'en og sammen med resten af systemet.

6.2.1 Pseudo Front-end Arkitektur

For at give overblik over, hvordan kommunikationen mellem frontend, backend og gamecontroller kommer til at foregå, er der lavet et pseudo sekvensdiagram for følgende UserStories:
- Login
- Register
- Save Game
- Load Game

Der vil i dette afsnit kun blive vist "Save Game". "Load Game","Login" og "Register" kan findes i Tekniskbilag **(INDSÆT REFERENCE HER)**.
På Figure 6 ses "Save Game", som viser forløbet når en bruger gerne vil gemme sit igangværende spil, set fra Frontends perspektiv. Her kan man bide mærke i, at når der skiftes skærm, vil den nye skærm få initialiseret sine variabler i sin constructor og derfor er der kun et selvkald, hver gang skærmen skiftes. Dette selvkald tager hånd om at opdatere mappen således at spilleren er i det rigtige rum, og at man kun kan se det af mappen, man har været i. Hertil skal der nævnes at hvis

brugeren er i "Combat State" er det ikke muligt at gemme spillet og knappen "Save Game" på "In Game Menu" vil ikke kunne ses eller bruges.

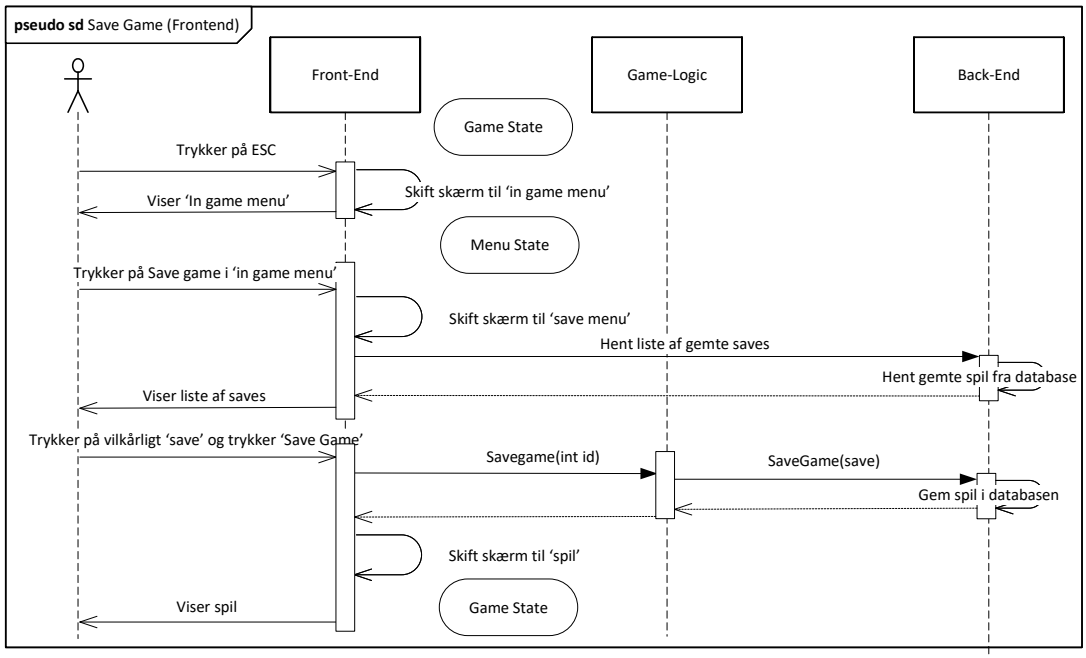

Figure 6: Pseudo sekvensdiagram af forløbet af userstory "Save Game", set fra Frontends perspektiv. Der laves 2 kald til databasen igennem Backenden, hvori der i det første kald, "Hent liste af gemte saves" hentes en liste af brugerens gemte spil og i andet kald gemmes brugerens nuværende spil henover det valgte spil.

6.3 GameController Arkitektur

Game Controllerns rolle er at skabe logikken for brugeren i spillet. Game Controllerens rolle for systemet er afgørende, når spillet er startet for brugeren. Game Controllerens funktionalitet indebærer blandt andet at skabe et map for brugeren, så spilleren kan flytte fra rum til rum ved start af spil. Her forekommer det at brugeren anvender Front End til at interagere med spillet og derefter kan funktionerne fra Game Controlleren kaldes.

6.3.1 States og Character interagering

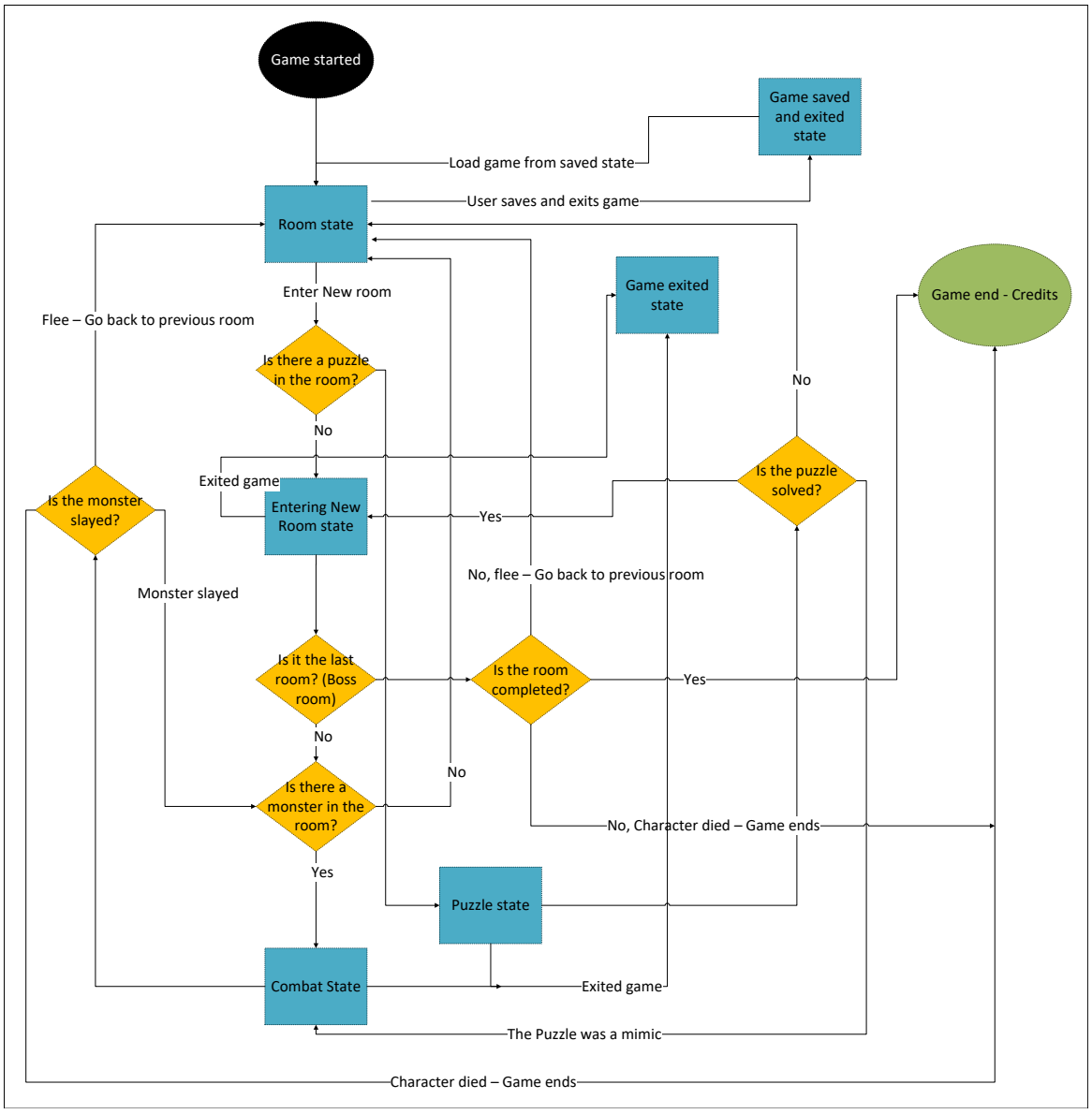

Figure 7: Flow Chart over systemets states når spillet er startet. Dette bilag viser arkitekturen over det ønskede states i systemet når bruger har startet spillet. Dette viser også hvordan spillets fremgang er ønsket og hvordan bruger kommer videre i spillet igennem de forskellige states.

6.3.2 Room State og Character interagering

Game Controlleren er delt op i spillets game states som kan ses i bilaget ovenover. Der er hovedsageligt 2 states når spillet er startet. Room State og Combat state. Når spillet er startet for brugeren, starter brugeren i et room state. I dette state kan der interageres med rummet, hvis der er genstande til stede. Her kan brugeren også brugerdefinere sin spiller ved brug af knappen inventory og skifte våben eller skjold. Her kan der også se spillerens evner ved brug af knappen Character. Derudover er der også implementeret beskrivelser fra de forskellige rum som hentes fra modulet Database via modulet Back end.

6.3.3 Combat State og Combat simulering

Combat state indebærer når spilleren møder en fjende. Dette sker når man går ind i forskellige rum. Der er oprettet klasser for spillerens evner i form af Attack(Spillerens evne til at slå) og Armor(Spillerens evne til at modstå angreb). Når spilleren angriber en fjende foregår det ved at brugeren trykker på knappen 'Attack'. Når dette forekommer er der implementeret en terning i Game Controlleren. Dette skal implementeres ved to implementeringer. En simuleret terning i form af en pseudo random-number generator som genererer et tal mellem 1 og parameteren numOfSides og en En Rekursive Psudo-Random number generator som tager en tuple (numOfSides, numOfDice). Den gentager rekursivt implementering 1. Et antal gange svarende til NumOfDice parameteren og summere alle resultaterne. Hvis brugeren taber kampen, skal brugeren starte et nyt spil eller loade et save. I figuren under ses et Flow chart over forløbet når spiller går ind i combat state.

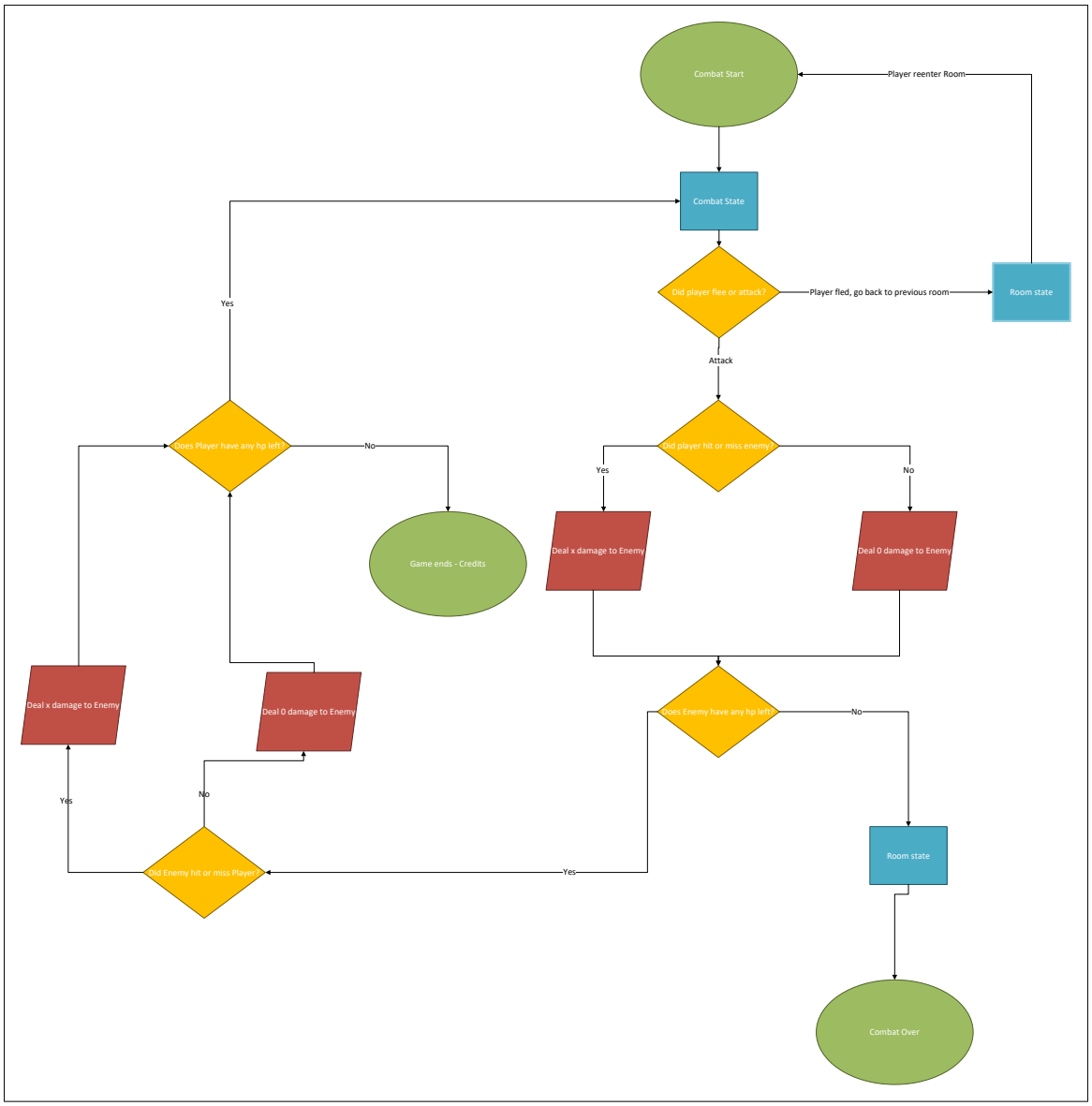

Figure 8: Flow Chart over spillets fremgang når spiller går ind i combat State. Når spiller går ind i et rum med en fjende i rummet, går spillet ind i combat state. Herfra skal spillets terning simulerer et tilfældigt nummer ud fra spilllerens evner. Fjendens evner er bestemt i forvejen. Angreb fastlægges om spiller ruller højere end fjendens Armor Class og vice versa De tre outcomes for spilleren er følgende: Spiller flygter(flee-knap), Spilleren mister alt liv(Spil sluttes) og fjende mister alt liv(Spiller går til room state)

6.4 Database Arkitektur

Systemets database vil stå for at opbevare data for brugerne, dette vil være i form af data indeholdt i et gemt spil.
I oprettelsen af systemets database skal der tages hånd om, hvordan kommunikationen skal foregå imellem systemets segmenter, samt databasens funktionalitet. Her er der blevet besluttet at der

anvendes en DAL, der fungerer ved at hver gang databasen skal kontaktes, så foregår det igennem denne. Yderligere vil denne DAL også simplificere kommunikationen mellem databasen og Back-end'en.
Herunder vises et eksempel på hvordan kommunikationen ville foregå med databasen.

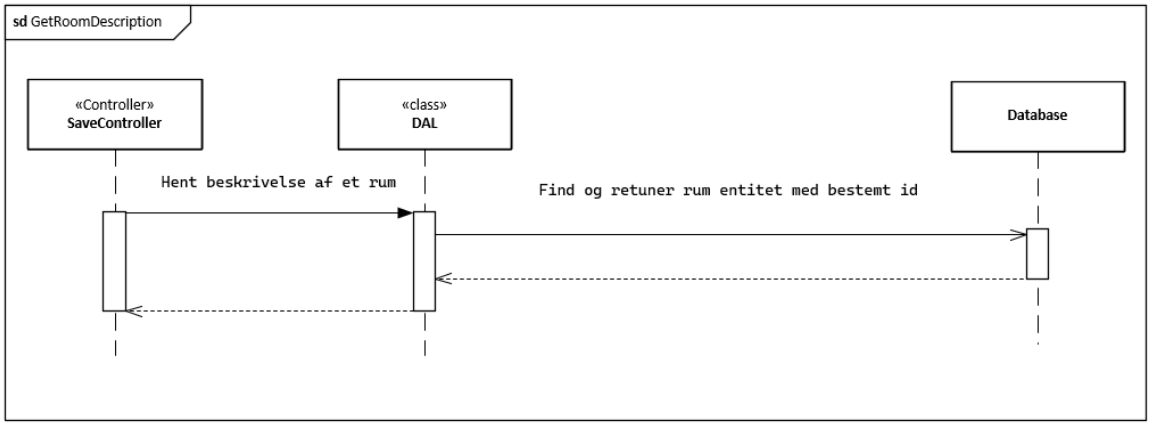

Figure 9: Sekvensdiagram for hvordan rum beskrivelser vil blive hente fra databasen af SaveController

I Figure 9 ses diagrammet GetRoomDescription ses der, hvordan kommunikationen ville foregå for at hente rum beskrivelsen. Her ses der at SaveController, fra Back-End, går til DAL, som så henter rum beskrivelsen fra databasen, og herefter returneres dataene fra databasen til DAL og så fra DAL til SaveController.
Yderligere eksempler og forklaringer vedr. kommunikation kan findes i teknisk bilag **MANGLER REF HER**

6.4.1 DAL Arkitektur

Når data enten skal sendes til eller hentes fra databasen så er det igennem et DAL. Systemets DAL fungerer som en mellemmand for systemet, da alt kommunikation til og fra databasen går igennem den.
I dette DAL vil data for at gemme et spil og indlæse et spil blive sendt igennem. Begge af disse vil indeholde det samme type data, dog ville den ene, LoadGame, hente data fra systemets database, og den anden, SaveGame, vil sende data til systemets database. LoadGame, og SaveGame, i DAL vil være ansvarlig for at hente spillerens data fra databasen, såsom hvilket rum de var i og mængden af liv de har tilbage.

7 Design

7.1 Overordnet System Design

Dette afsnit repræsenterer hvordan modulerne ønskes at kommunikere med hinanden, samt forsøger at give et overblik over, hvor meget der kommunikeres mellem modulerne i de forskellige stadier. Der vil i dette afsnit indgå 2 User stories til at repræsentere systemets design. Disse User Stories er relevante for design, da de involverer alle moduler samtidigt og giver et indblik i kommunikationen på tværs af systemet. For at se flere User Stories og hvordan de vil virke i systemet se teknisk bilag **Indsæt reference her**.

Herunder ses et sekvensdiagram for User Story 15 og 18, nemlig Save og Load game. Her ses hvordan det ønskes at de forskellige moduler kommunikerer på tværs af systemet, når en bruger at hente eller gemme data fra/i databasen.

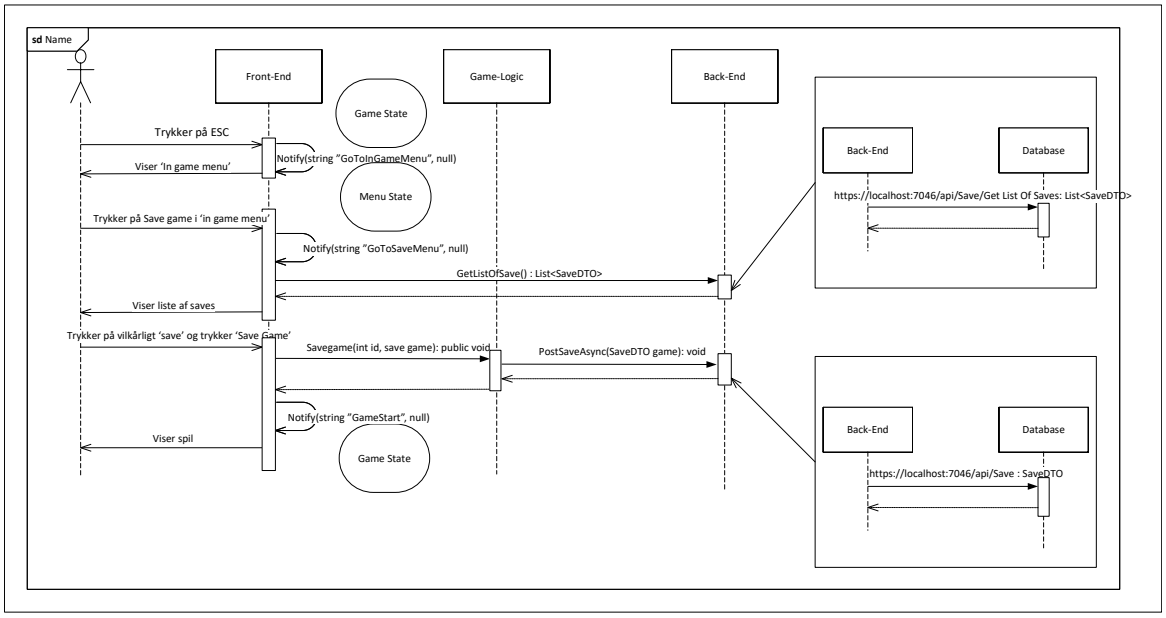

Figure 10: SD diagram for User Story 15 " Save Game". Diagrammet viser hvordan det ønskes at systemet overordnet skal kommunikere på tværs af hinanden når en bruger skal gemme sit aktuelle save

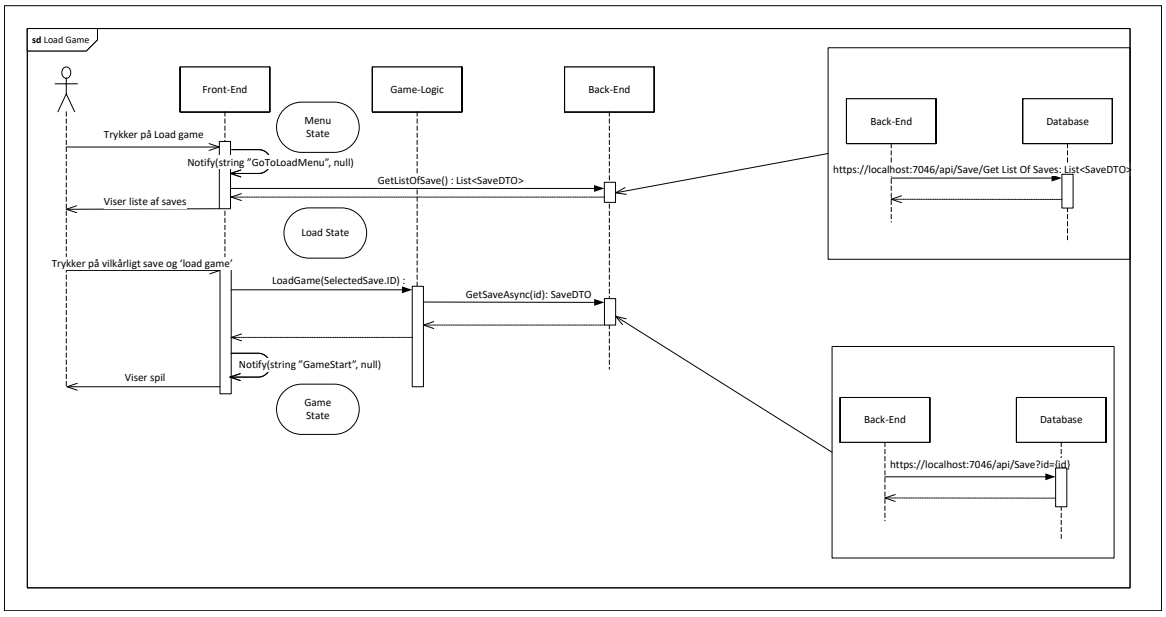

Figure 11: SD diagram for User Story 18 "Load Game". Diagrammet viser hvordan systemet overordnet skal kommunikere på tværs når bruger skal loade sit aktuelle save.

7.2 Front-end Design

Inden arbejdet på Front-end arkitekturen begyndte, er der blevet lavet en teknologiundersøgelse **??**, om hvilket udviklingsværktøj Front-enden og derigennem spillet skulle udvikles i. Baseret på denne teknologiundersøgelse er der blevet valgt, at spillet vil blive udviklet i et .NET framework. Dette valg er blandt andet truffet, da dette framework passer bedre med den opdeling af arbejde der er lavet i projektgruppen, altså opdelingen af Frontend og Backend. For andre grunde, se **??**, hvor flere fordele og ulemper for både unity og .NET frameworket er sat op.

Spillets Front-end består af en række menuer, samt selve spil viewet. Spillet skifter så mellem disse forskellige menuer og views i samme vindue, således at brugeren får en flydende overgang. Det primære spil-vindue er room view, et mock-up af dette view kan se på Figure 12. Her præsenteres spilleren for en beskrivelse af det rum de er i, samt hvilke elementer i rummet de kan interagere med. Der vises også et kort over banen. Kortet viser kun de rum spilleren allerede har været i, mens resten holdes skjult. Når brugeren så besøger et nyt rum, kan dette ses selvom spilleren forlader rummet. Dette lader spilleren udforske og oplåse hele kortet.
En række knapper nederst i højre hjørne på skærmen giver spilleren mulighed for at interagere med spillet. Fire knapper ("Go North/West/South/East") lader spilleren gå fra et rum til et andet. Ikke alle rum har forbindelse til alle sider, så det er f.eks. ikke altid muligt at trykke på "Go North". Kortet og rum beskrivelsen fortæller hvilken vej det er muligt at bevæge sig i.
Udover de fire retningsknapper er der et antal andre knapper. Disse bruges til at gemme spillet, gå til menuer, samt interagere med elementerne i rummet. Det specifikke antal og deres funktion er afhængig af den præcise implementering.

For at læse mere om spillets andre menuer og views se **INDSÆT REFERENCE HER**

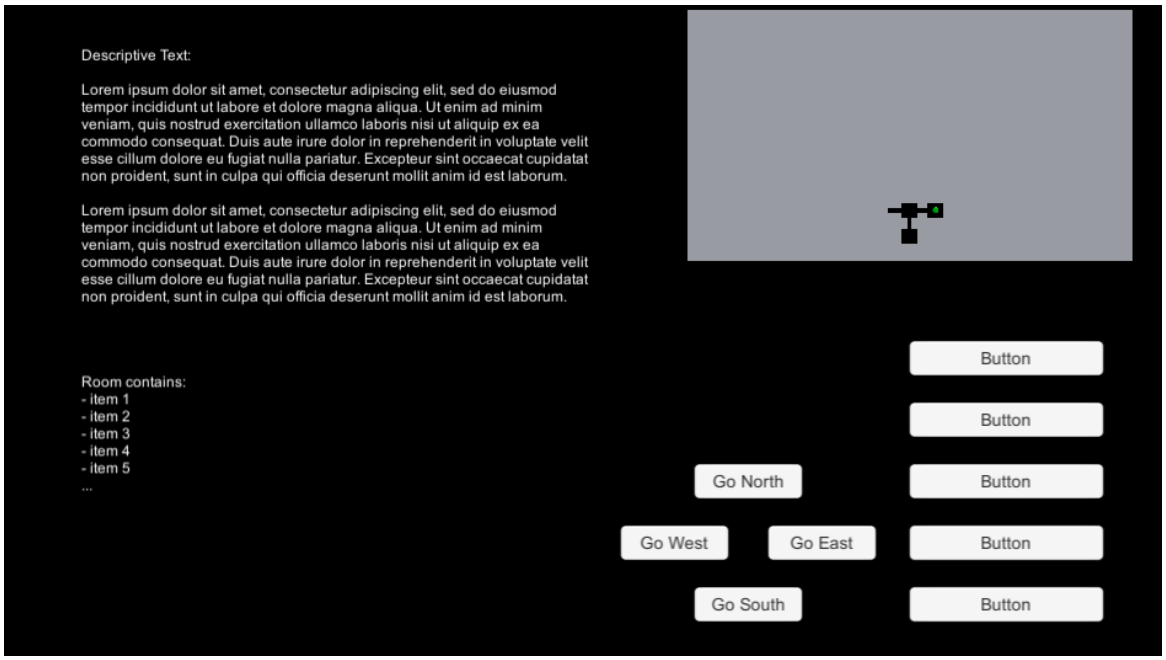

Figure 12: Et mockup af det primære spil vindue. Tekst øverst i venstre side af skærmen giver en beskrivelse af det rum spilleren er i, samt en liste af elementer i rummet som spilleren kan interagere med. Øverst til højre vises et billede af banen. Spilleren interagerer med spillet via knapper nederst i højre hjørne. Knapperne "Go North/West/South/East" fører spilleren ind i et andet rum, mens de resterende knapper (markeret "Button") bruges til andre funktionaliterer i spillet.

7.3 Game Engine

Spillets logik styrres fra game engine, der består af de komponenter, som modellere spillets verden og dens logik. Af disse komponenter er Game Controlleren, Combat Controlleren, Room, og Player de mest essentielle.

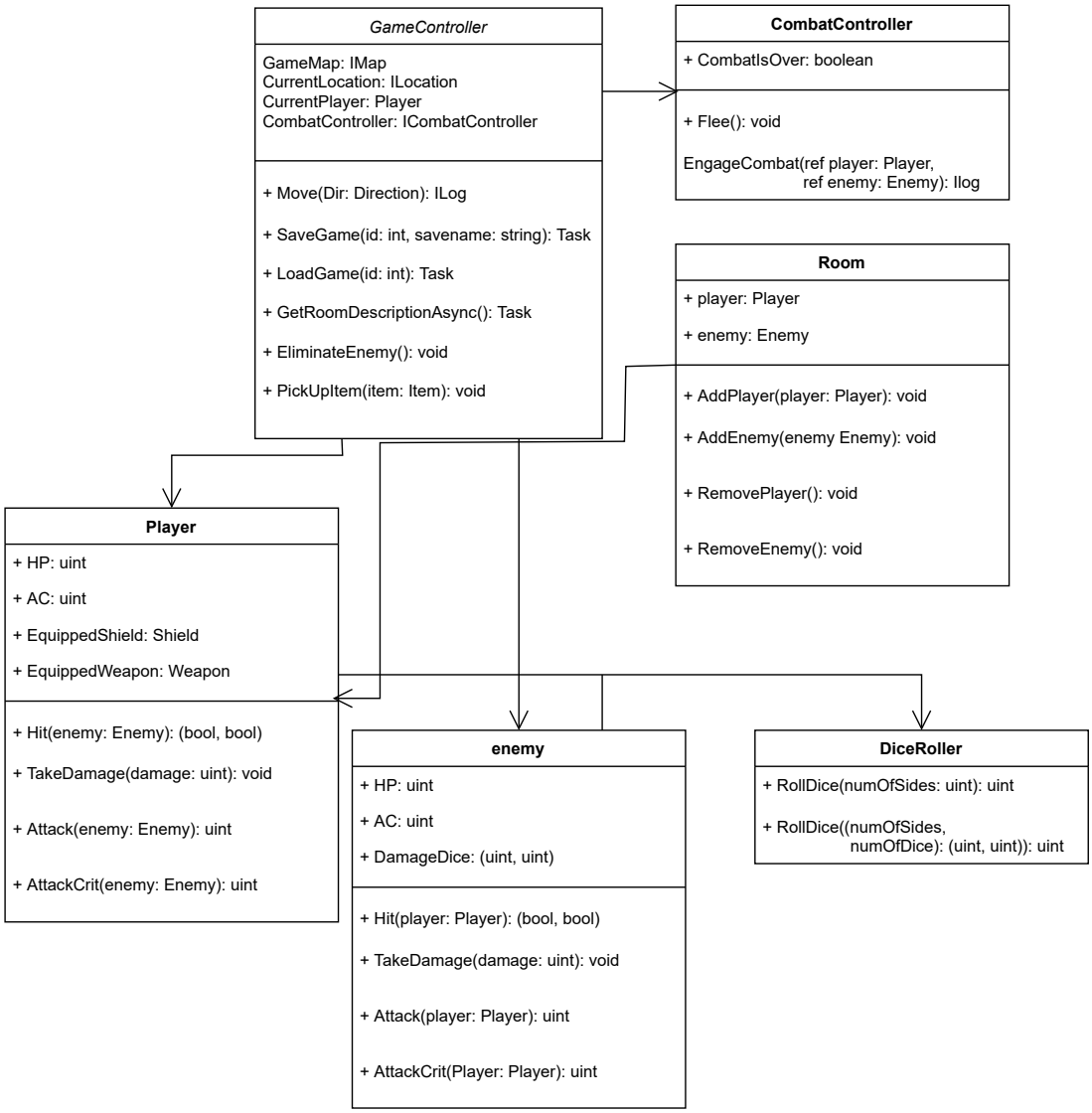

Figure 13: De vigtigste elementer af game engine og deres relationer til hinanden. Der er her ikke vist interfaces, men lagt fokus på de kritiske elementer som de er implementeret.

7.3.1 Game Controller

Game Controlleren styre alle UI funktionaliteter, når brugeren trykker på en knap, der påvirker spillets state, gøres dette gennem game controlleren. Game controlleren instantieres når et spil startes; hvilket danner et map [Section 10.3.1 2], spilleren kan navigere rundt på.
Game controlleren tillader at spilleren kan bevæge sig mellem spillets forskellige rum [Section 9.3.1 2] i overenstemmelse med spillets map layout. Skulle spilleren forsøge at bevæge sig i en retning, der ikke er tilladt ifølge map layoutet bliver spilleren stående i det nuværende rum.
Skulle spilleren bevæge sig ind i et rum med et "item" styre game controlleren spillerens evne til at inteagere med "itemet". Spilleren har evenen til at samle "items" op, og derved øge deres stats.
I det tilfælde, at spilleren bevæger sig ind i et rum med en modstander, giver game controlleren spilleren evnen til at bekæmpe fjenden eller flygte fra denne. Game controlleren gør dette ved at

kalde combat controlleren, som håndtere kamp i spillet.

7.3.2 Combat Controller

Combat controlleren håndtere spillet logik, i det omfang det relatere sig til kamp mellem spilleren og en fjende [Section 9.3.3 2, Figur 17].
Combat i spillet afvikles ved hjælp af flere simulerede terningekast (RNG) se dicerollern, i Figure 13 der afgører både om spilleren/fjenden rammer fjenden/spilleren og hvor meget skade fjende/spilleren modtager. Spillets "items" har indflydelse på vægtningen af disse simulerede terningekast, og kan hjælpe spilleren med at ramme og skade modstanderen. "items" kan også gøre det svære for fjenden at ramme spilleren.
Spilleren kan efter hvert sæt af terningekast vælge at forsætte kampen eller flygte. Skulle spillerens liv nå nul, dør spilleren og spillet er færdigt.

7.3.3 Room

Et "Room" eller rum er en diskret delmængde af spillets verden, som kan indeholde spilleren, fjender og forskellige genstande. "Room" er kun ansvarlig for at tilføje og fjerne spilleren og fjender for rummet se Figure 13. Ikke desto mindre kunne spilleren ikke navigere verden uden existensen af disse diskrete områder.

7.3.4 Player

Spilleren karakter (PC) repræsentere spilleren i spillet. Dette komponent er ansvarlig for at holde styr på spillerens tilstand i spillet og giver spilleren evnen til at forsvare sig selv i kamp. Denne simulere en spillers forsøg på at ramme en fjende, skade en fjende og opdatere spilleren liv, forsvar og "items" undervejs i spillet [Section 13.3.2 2].

7.4 Backend Design

Design overvejelser angående systemets backend Web Api vil her blive gennemgået. Først præsenteres hvilke routes Api'et stiller til rådighed. Herefter gennemgåes hvordan Api'et håndtere authentication/authorization, samt hvordan passwords vil blive gemt. Hertil vil en BackEndController klasse, som skal bruges på client siden blive redegjort for. Tilslut præsenteres resultatet applikations modellen nemlig klasse digrammer for de to controllers udførelse af User Stories.
For ydderligere detlajer omkring designet af backend'ens Web Api henvises til bilag **Mangler ref til bilag bed design**
I forbindelse med udviklingen af Backendens Web Api er der lavet en tekniskanalyse som kan findes i afsnit (**ref til tekanalyse**), konklusionen her på blev at anvende frameworket ASP.Net Core.

7.4.1 Routes

Applikationens nødvendige routes kan inddeles i to under grupper "Save" og "User". En oversigt over de forskellige routes med en tilhørrende beskrivelse kan ses nedenfor.

Save:

- GET: /api/Save Denne route henter et sepcifikt gemt game state fra databasen, for den bruger som er logget ind.
- POST: /api/Save
  Denne route sender et scecifikt game state til databasen, for den bruger som er logget ind.
- GET: /api/Save/Get List Of Saves
  Denne route henter en liste af game states, for den bruger som er logget ind.
- GET: /api/Save/Get Room Description
  Denne route henter en beskrivelse af det valgt rum i spillet.

User:

- POST: /api/User/Register
  Denne route registrerer en ny bruger ved at gemme oplysninger om denne i databasen, og returnerer en JWT token. Når en bruger bliver registreret får den tildelt 5 pladser i databasen til game states.
- POST: /api/User/Login
  Denne route logger en bruger ind ved at tjekke bruger oplysninger med dem, som er registreret i databasen, denne route returnerer også en JWT token.

7.4.2 Authentication/Authorization med JWT Token

Til at validere og afgøre om en bruger har adgang til en given resource anvendes JWT (Jason Web Token). Til hashing af JWT'en anvendes algoritmen "HmacSha256". Af private Claims benyttes brugerens Username, da dette er unikt og kan afgøre om brugeren har ret til det aktuelle spil

For at generere denne token implementeres en funktion, som opretter en token efter de forhold som er beskrevet ovenfor.

7.4.3 Hashing

Da en brugers password er følsom data, krypteres dette i backend'en igennem hashing inden det gemmes i databasen. Til at udføre hashing anvendes Bcrypt [**Bcrypt**].

7.4.4 BackEndController på client siden

Til brug af clienten for at tilgå backenden, skal der bruges en BackEndController klasse, dennes ansvar vil være at udføre HTTP request/responses, og vil indeholde en funktion for hvert endpoint. Klassen vil gøre brug af en HttpClient som er en klasse .Net stiller til rådighed til at håndtering af HTTP request/responses.

7.5 Applikations modeller

Der udarbejdes appilationsmodeller for de to controller klassers håndtering af User stories, de frembragte klasse diagrammer kan ses nedenfor.
Usercontrolleren har til opgave at gøre det muligt for en bruger at registrere sig og logge ind. Et klasse diagram som viser de nødvendige funktioner for udførelsen af User story 1 og 2 kan ses på Figure 14

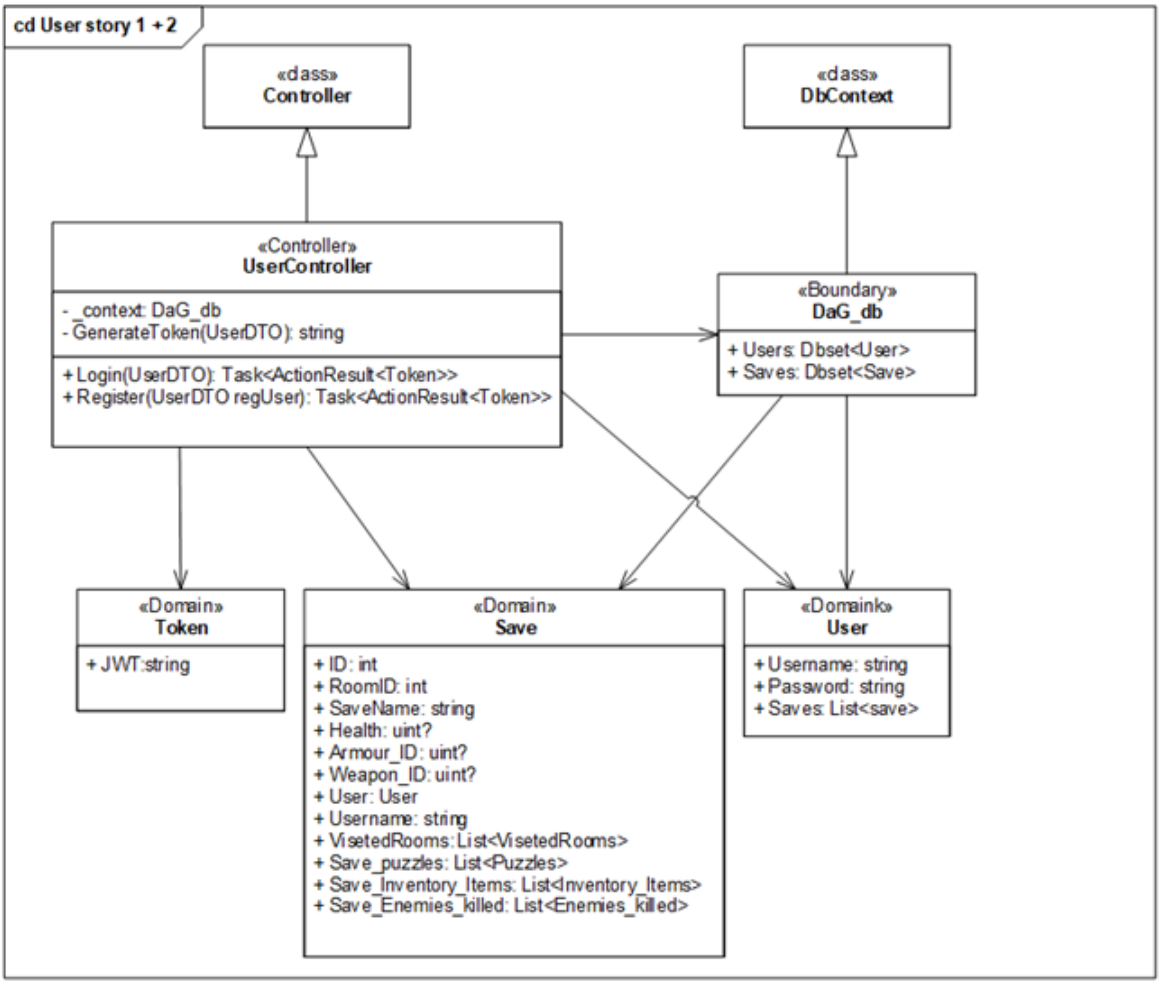

Figure 14: Samlet klasse diagram over User story 1 og 2

SaveControlleren har til ansvar at gemme og hente spil. Et klasse diagram for dennes håndtering af User Story 15,17 og 18 kan ses på Figure 15.

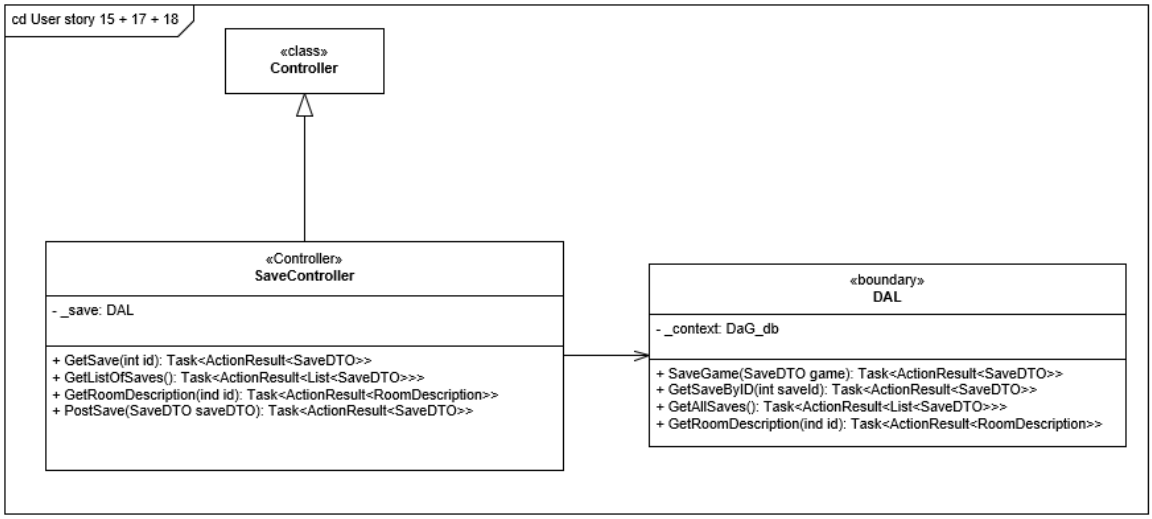

Figure 15: Samlet klasse diagram for User story 15, 17 og 18

En fuld applikationsmodel over UserController og SaveControllers håndtering af de relevante User Stories kan ses i bilag **ref til appmodel**.

7.5.1 Konklusion

Backenden er blevet designet til at stille authentication/authorization til rådighed igennem, JWT. Derudover er der blevet opstillet de nødvendige routes for udførelsen af User Stories, som er blevet fundet igennem applikationsmodeller. BackendControlleren klassen blevet introduceret på clienten, som står for HTTP request/response.

7.6 Software Design DAL Design

På Figure 16 herunder ses klassediagrammet over backend DAL, som benyttes til database access. Som det kan ses på diagrammet, indeholder DAL en database context, som benyttes til at forbinde backend applicationen til databasen. Derudover består DAL af fire overordnede funktioner, hvoraf de 3 tilhører en user story. Den sidste funktion benyttes når vi starter spillet, til at lave rumbeskrivelser.

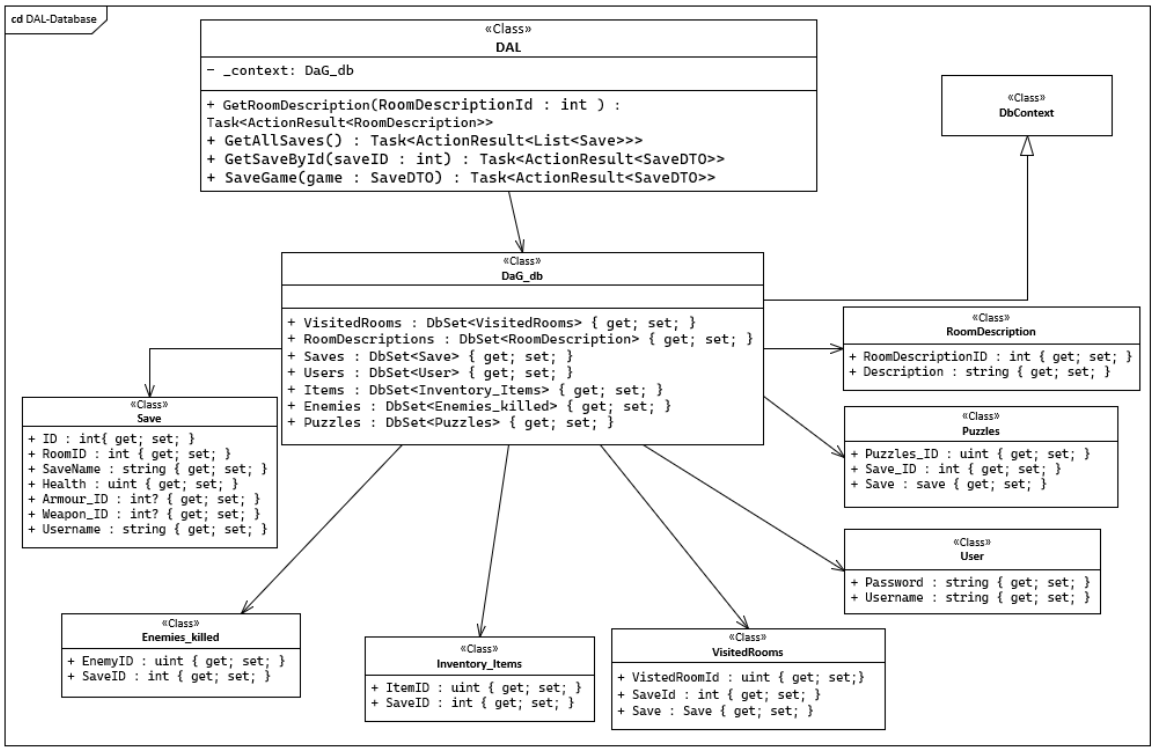

Figure 16: Samlet klasse diagram for User story 8, 15, 17 og 18 med DAL db context og entitetsklasser. For læseligheden er DAL klassen ikke forbundet til forskellige entitetsklasser selvom disse benyttes.

Følgende funktion som beskrives på Figure 17 benyttes når spil klienten startes, da beskrivelser af rum ikke ændre sig gennem spillets levetid, i den nuværende itteration.

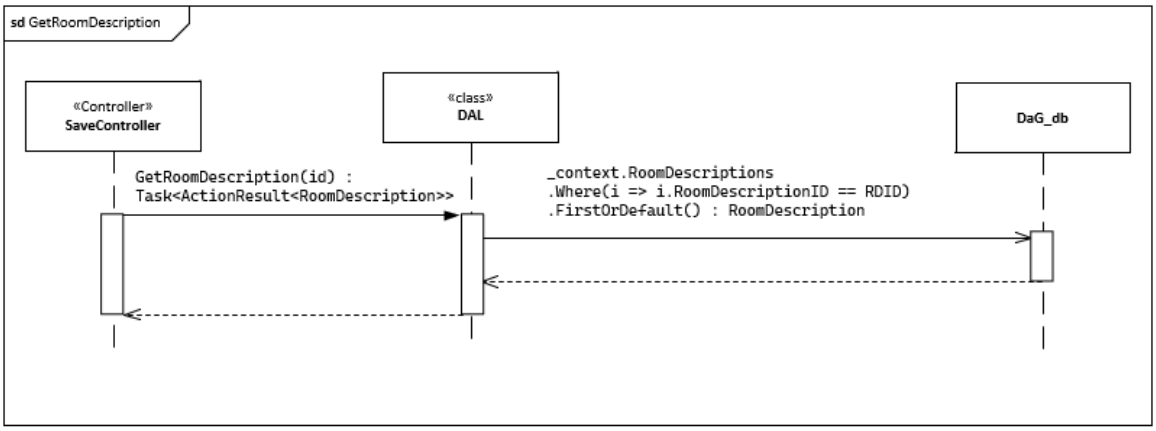

Figure 17: Sekvensdiagram om læsning af en rumbeskrivelse

Navn: GetRoomDescription
Parametre: int RoomDescriptionId
Returtype: Task<ActionResult<RoomDescription>>
Beskrivelse: Denne funktion finder og returnerer en beskrivelse for det valgte RoomDescriptionID. Dette kan også ses på sekvensdiagrammet på figur **REF HERE** herunder.

7.6.1 User story funktioner

De følgende 3 funktioner benyttes til udførelse af forskellige user stories. Det drejer sig om:

- User story 15 – Save game
- User story 8 - Save - No Combat + User story 17 – Load game list
- User story 18 – Load game

Den første funktion SaveGame benyttes til at udføre user story 15. Her skal der gemmes et spil. Da vi har et krav om kun at holde 5 gemte spil pr. bruger, vælger vi at oprette 5 "tomme" saves, når vi opretter brugeren. Når brugeren så ønsker at gemme et spil, skal vi ikke tilføje et nyt, men istedet overskrive et valgt gammelt save. Dette kan også ses på sekvensdiagrammet på figur Figure 18

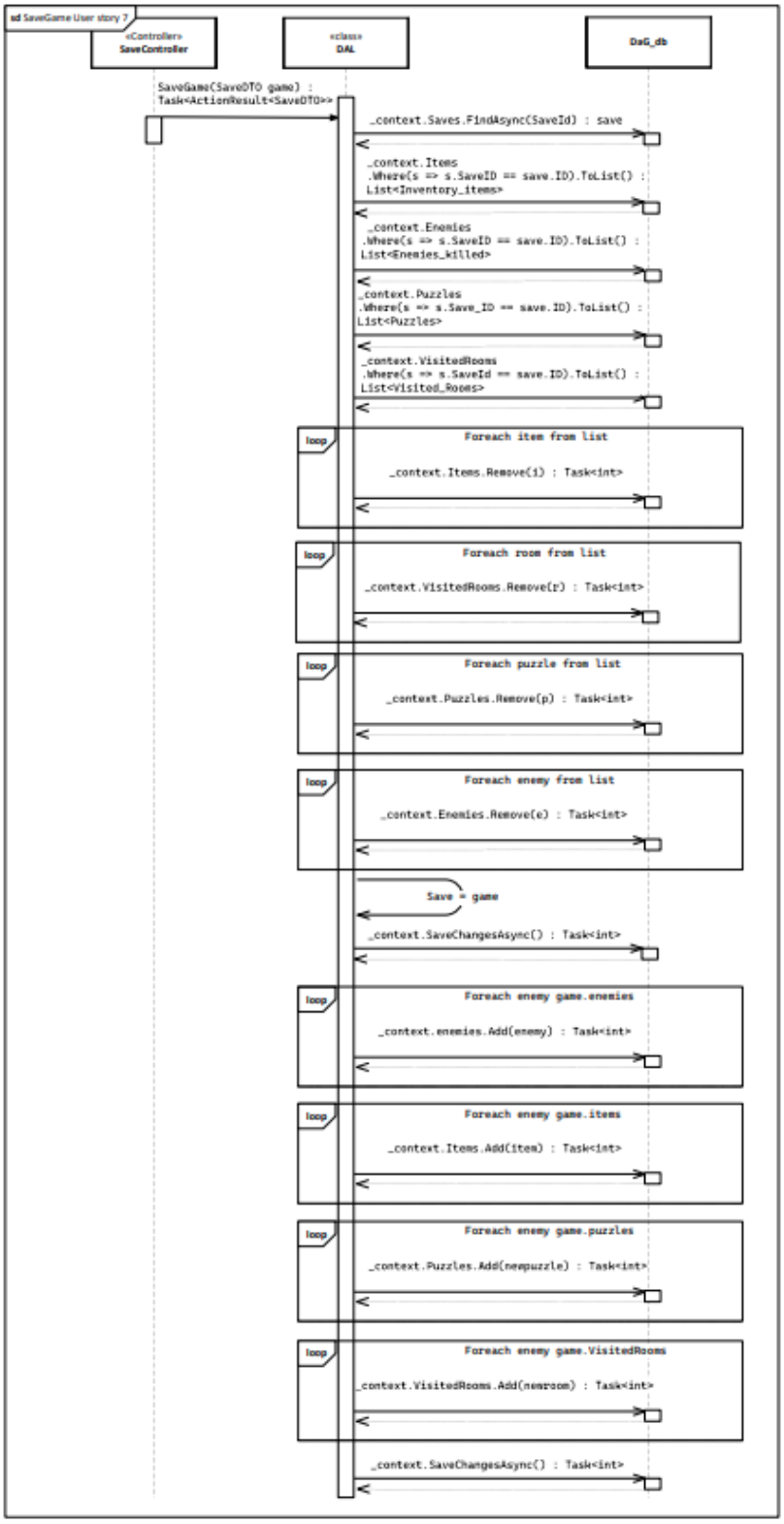

Figure 18: Sekvensdiagram for user story 15 save game som beskriver queries til databasen for at gemme et spil

Navn: SaveGame
Parametre: SaveDTO
Returtype: Task<ActionResult<SaveDTO>>
Beskrivelse: Da der i vores frontend sørges for at en bruger blot kan have 5 saves, starter vi med at finde det save vi gerne vil overskrive. Det gamle save, samt tilhørende lister slettes, hvorefter det nye save gemmes og eventuelle nye lister til fx. items gemmes.

Den anden funktion GetAllSaves benyttes til at udføre user story 8 og 17. Her skal der loades en liste af gemte spil, når brugeren ønsker at se sine gemte spil. Dette kan også ses på sekvensdiagrammet på Figure 19 .

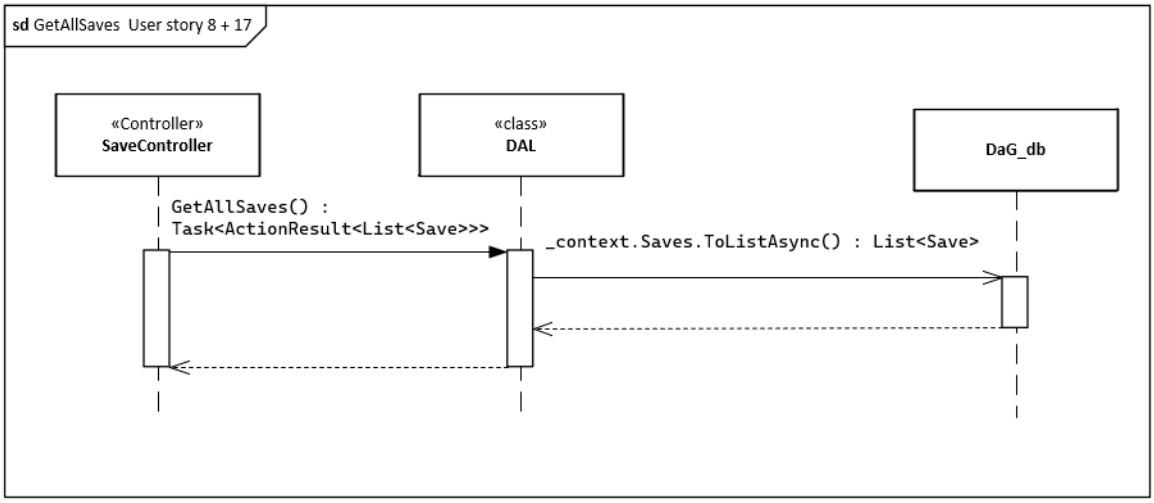

Figure 19: Sekvensdiagram for user story 8 og 17 GetAllSaves som beskriver queries til databasen for at hente alle saves

Navn: GetAllSaves
Parametre: ingen
Returtype: Task<ActionResult<List<Save>>>
Beskrivelse: Her hentes all gemte saves i spillet.

Den tredje funktion GetById benyttes til at udfører user story 18. Her skal der loades et gemt spil, som brugeren nu ønsker at spille. Dette kan også ses på sekvensdiagrammet på Figure 20.

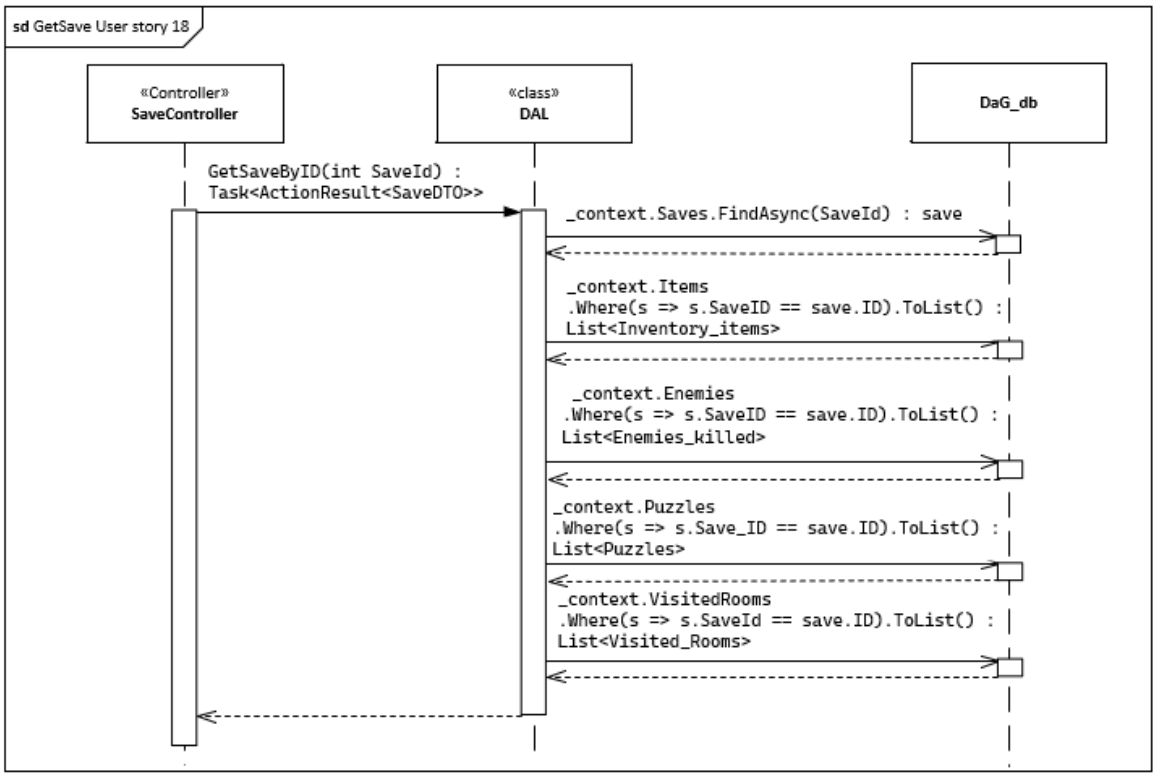

Figure 20: Sekvensdiagram for user story 18 GetSaveById som beskriver queries til databasen for at hente et specifikt save og dens tilhørende information

Navn: GetSaveById
Parametre: int saveID
Returtype: Task<ActionResult<SaveDTO>>
Beskrivelse: Denne funktion finder det save med det medsendte ID, samt tilhørende lister, indsætter værdier i et SaveDTO objekt, hvorefter det returneres.

7.7 Database Design

Projektets database har gruppen valgt at hoste i lokal storage. Dette er valgt da der under semesterets forløbet opstod problemer med skolens licens af Microsoft produkter. For at undgå at komme ud for udfordringer med hosting senere i forløbet, blev der valgt at gå med den sikre løsning, at hoste databasen lokalt på enheden. Til dette benyttede gruppen et docker image[**SQL server with docker**], specifikt det samme image som blev benyttet i DAB undervisning, til vores SQL server. Hvis ikke der var problemer microsoft, havde gruppen i stedet valgt at lagre data på en cloud-based storage fremfor lokal storage.

For at kunne udarbejde et ER diagram til modellering af vores sql database skal vi start med at finde ud af hvilke krav vi har til og hvilke attributter vi ønsker at gemme i vores database. Først og fremmest ønskede gruppen at vi kunne gemme beskrivelserne af de forskellige rum, i spillets layout, for at formindske antallet af filer i klienten, og samtidigt gøre eventuelle senere tilføjelser nemmere. Her benyttes rummets id som key, da vi ikke ønsker at man skal kunne oprette flere beskrivelser til samme rum. Diagrammet kan ses på Figure 21.

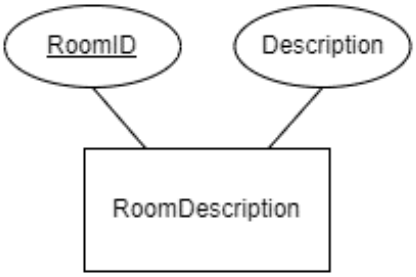

Figure 21: ER diagram for Roomdescription. En beskrivelse består blot af en beskrivende string samt det tilhørende unikke rumid.

Her efter kommer kravene til at kunne gemme et spil for en bruger. Her ønskede vi at man kunne stå et vilkårligt sted i spillet, med untagelse af en combat, og gemme spillet. Det skulle derefter være muligt for spilleren at loade spillet igen, hvorefter spillet er i samme stadie som man gemte det i. ER diagrammet for at gemme et spil til en bruger på Figure 22.

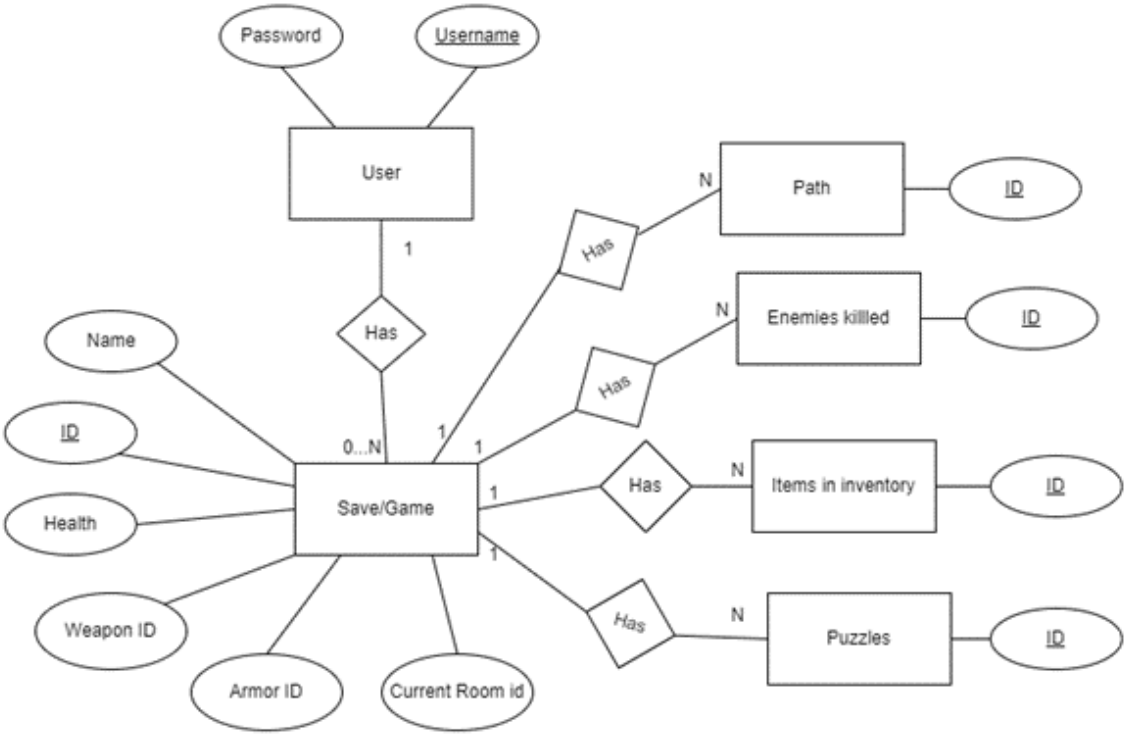

Figure 22: ER Diagram til at gemme et spil til en specifik bruger. Her ses de forskellige informationer som skal til for at kunne gemme et helt spil

Først og fremmest ønskede gruppen et bruger system, så eventuelle gemte spil kun tilhørte en bruger. Der gemmes derfor en bruger entitet med et unikt brugernavn og et tilhørende password. Sikkerhed på password og versalfølsomhed på brugernavnet håndteres af spillets backend.
En spiller skal derefter kunne gemme 5 unikke spil med forskellige oplysninger. Restriktionen med max 5 forskellige spil pr. Bruger, håndteres ved at oprette 5 "tomme" gemte spil ved oprettelsen af

en bruger. Et af disse gemte spil overskriver derfor når vi gemmer. Der kan på denne måde ikke oprettes mere en 5 gemte spil pr. bruger.
I et gemt spil ønkede vi at gemme en række forskellige attributter for spilleren. Første og fremmest får hvert spil et unikt id som vi benytter til identifikation og at lave forhold mellem de forskellige tabeller. Et gemt spil får et navn, valgt af brugeren, som gør det nemmere for brugeren at differentiere mellem de forskellige spil.
Dette navn skal være forskelligt fra de 4 andre gemte spil som tilhører samme bruger.
En spillers Health gemmes også, da man kan have taget skade efter en kamp.
Det gemmes også hvilket rum, spilleren står i når spillet gemmes, så vi loader korrekt tilbage. En spiller kan derudover også holde genstande, som armor og våben, i hånden eller i sit inventory. Dette gemmes også henholdsvist som en del af et spil og i en inventory liste tilhørende spillet.
Tabellen med inventory har 2 atributter, et ID, som svarer til en bestemt genstand, og en reference til et SaveID. Denne parring er unik, da man ikke kan holde 2 af den samme genstand.
Tabellerne med Enemies og puzzles fungerer på samme måde. Her har hver enemy og puzzle i spillet et unikt id. Id'et gemmes i kombination med et saveId, som et unikt par, da man ikke kan vinde over samme enemy og løse samme puzzle flere gange.
Til slut ønskede vi at kunne vise spilleren de rum som allerede er blevet besøgt. Derfor gemmes der i path tabellen, for hvert spil, en unik kombination af saveID og besøgte rum id. Denne parring er unik da man blot behøver at besøge et rum en gang, før det er synligt på kortet.
Der er i projektet oprettet klasser tilsvarende ER diagrammerne. Forholdene mellem disse, samt keys, er opsat i DaG-db klassen og er skrevet ved hjælp af fluentAPI. Dette kan ses i Implementationsafsnittet subsection 8.5.

8 Implementering

8.1 System Implementering

Under implementeringen af nogle funktioner i alle 4 moduler, er funktioner blevet lavet til funktioner der følger følgende opstilling:
public async Task <T> Foobar();
Funktionerne returnerer en Task af typen T og dette kan gøres asynkront. Dette er specielt vigtigt når man kontakter databasen, da programmet ikke må køre videre før den asynkrone funktion er færdig med sit database kald. Når en funktion i f.eks. Game Controlleren kalder: public async Task <SaveDTO> GetSaveAsync(int id) i backend controlleren, i sin egen funktion SaveGame() skal SaveGame() også erklæres async og returnerer en Task af typen T og derfor er nærmest alle funktioner der kontakter databasen erklæret for async kald, som returnerer typen Task.

8.2 Front-end Implementering

Til Front-end implementering er der brugt WPF med MVVM design patterns. Til at skifte views uden at oprette et nyt vindue bruges et mediator design, hvor hver knap som skifter view giver en besked til mediatoren. Dette virker fordi alle views er oprettet som WPF user controls, og ikke views, hvilket tillader at et main view kan skifte mellem viewmodels, derved skifter vil alt indhold på vinduet, men uden at selve rammen skifter. Dette resulterer i et mere flydende User Interface for brugeren og derved en alt i alt bedre oplevelse. [1]
Et eksempel på hvordan mediatoren kaldes ved et tryk på en knap kan ses i følgende kode eksempel:

```
private DelegateCommand _loadGame;

public DelegateCommand LoadGame => _loadGame ?? (
        _loadGame = new DelegateCommand(
        ExecuteLoadCommand, CanExecuteLoadCommand));
```

```
async void ExecuteLoadCommand()
{
    await GameController.Instance.LoadGame(SelectedSave.ID);
    Mediator.Notify("GameStart", "");
}

bool CanExecuteLoadCommand()
{
    return SelectedSave != null;
}

void LoadCommand()
{
    LoadGame.RaiseCanExecuteChanged();
}
```

Visuelt er alle views implementeret tæt på det oprindelige design. Antallet og placering af knapper har ændret sig en smule, men den primære ide er uændret. Et eksempel på dette er room view. For at se mere om hvordan andre views og menuer er implementeret se Tekniskbilag **INDSÆT REFERENCE HER**.

8.2.1 Room View

Visuelt er room view (Figure 23) ikke ændret betydeligt fra det oprindelige design. Der er ændret lidt på placering og antal af knapper så det passer til antallet af interaktioner tilgængelig til brugeren. Kortet er lavet så det opdateres når spilleren går ind i et nyt rum, ved at ændre på synligheden af elementerne i kortet. Det er yderligere sat op så det kan skaleres til de skærmopløsninger som understøttes.
Ved at trykke på interact knappen kan spilleren flytte et valgt 'item' fra listen nederst til venstre over i sit inventory (et seperat view), som kan tilgås ved at trykke på Inventory knappen.
Alt tekst er vist med data binding.

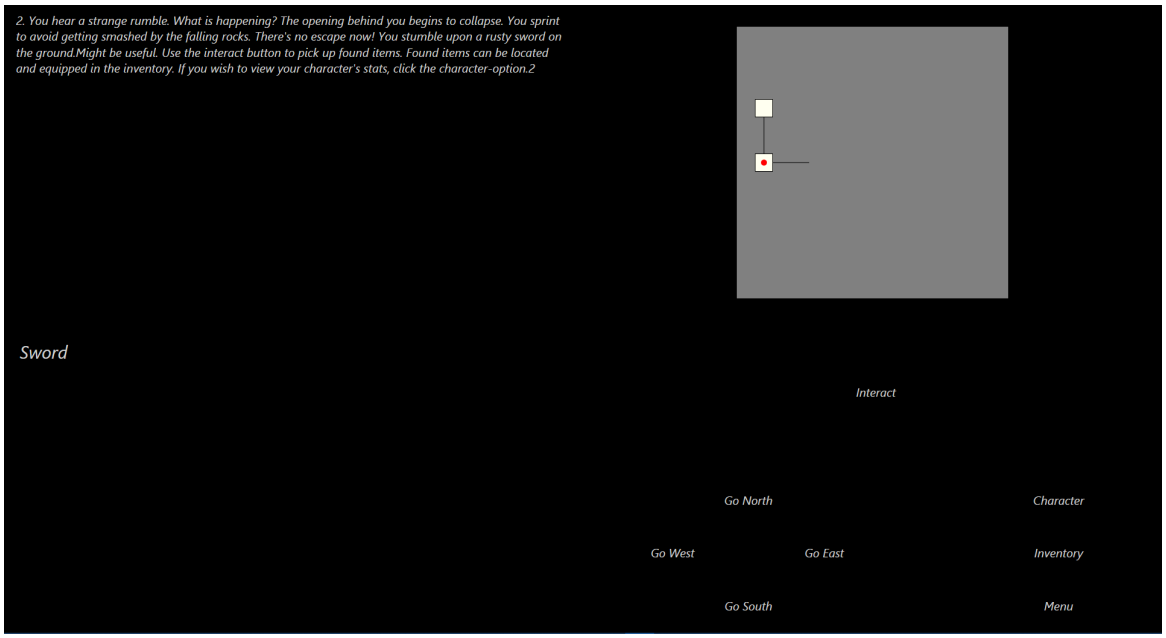

Figure 23: Endeligt udseende af room view. Generelt er der ikke ændret meget i forhold til det oprindelige design. Kortet er lavet så det skalerer med skærmopløsningen.

8.3 Game Engine

Game Engine er implementeret i C#, et objectorienteret programmeringsprog med stærk library support, der tillader udvikling af applikationer. Figure 13 giver en klar indsigt i hvordan kernedelen af game engine er designet til at virke.

8.3.1 Game Controller

Game controlleren er det centrale komponent i game engine; ansvarlig for kommunikation med frontend modulet. Game controlleren har adgang til alle aspekter af spillet, gennem sin association med Combat controlleren og backend controlleren.
Det er en nem løsning, og den fungerer godt givet det begrænsede scope som game enginen skal fungere under. For en mere kompliceret applikation med forventninger om udvidelser er det dog ikke en god implementering da game controlleren har for mange grunde til at ændre sig. Den har altså et for bredt ansvarsområde og kan formentlig splittes op i flere klasser.
Når game controlleren instantieres laves et map af Map og Map creator klassen hvilket før til en længere process hvor map creator klassen laver nogle layout filer som map klassen kan læse for at danne et map i spillet, som spilleren kan navigere [Section 11.3.2 2].

Game controlleren er i stand til at save og load games gennem dens relation til backend controlleren, der kan lave fetch til spillets database.
Selve Load process er kompliceret, da den involvere adskillige iterationer igennem alle rooms, items og fjender i spillet for at bekræfte om disse er blevet besøgt, indsamlet eller beseret tidligere [Section 11.3.1 2, Figure 55]. Save game er simplere, da game controlleren selv har adgang til alt infomation som er nødvendigt for et komplet save game.

8.3.2 Combat Controller

Skulle spilleren befinde sig i combat, er det combat controlleren, der styre programmets "flow of execution". Først angriber spilleren; fjenden hvorefter fjenden angriber spilleren hvis og kun hvis fjenden stadig er i live. Combat controlleren benytter en diceRoller til at simulere terningekast, der benyttes til at bedømme om spilleren og fjenden rammer med deres repektive angreb og efterfølgende; hvor meget deres angreb skader. Spilleren kan opsamle og benytte våben til at forbedre deres terningekast [Section 10.3.2 2].

8.3.3 Log

Game Controlleren kommunikerer med front-end gennem en log. Loggen logger forskellige events når game controlleren ændre game statet. Frontend kan derefter benytte loggen til at læse og vise hvilket ændringer, der er foretaget og præsentere dem til spilleren på en brugbar måde og brugervenlig måde.

8.4 Backend Implementering

I det følgende afsnit gennemgåes implementerings detaljer omkring Web api'ets controller klasser, samt for BackEndController klassen på client siden.
Alle controller klasserne er implementeret i C#, da dette er sproget som anvendes i Asp.net Core.

8.4.1 SaveController

SaveController klassen består af en række HTTP metoder, disse defineres ved at anvende atributter fra librariet Mircosoft.AspNetCore.MVC, som eksempelvis HttpGet se Figure 24, der viser implementeringen af GetSave.

```
// GET: Save
[HttpGet]

0 references | suiunicui, 4 days ago | 2 authors, 4 changes
public async Task<ActionResult<SaveDTO>> GetSave(int id)
{
    var identity = User.Identity;


    var save = await _save.GetSaveByID(id);

    if (save == null)
    {
        return NotFound();
    }

    if (identity.Name == save.Value.Username.ToLower())
    {
        return save;
    }
    else
        return Unauthorized();

}
```

Figure 24: Code snippet af HttpGet metode som modtager et id og retunere et SaveDTO object til clienten

På figuren ses en implementering af http metoden som bruges til sende et gemt spil til clienten. Funkionen retunere en Task ActionResult, som kan indeholde et SaveDTO object eller blot en status kode.
SaveController klassen tillader kun kald fra en korrekt bruger, derfor tilføjes atributten Authorize fra Microsoft.AspNetCore.Authorization librariet. User.Identity benyttes så inde i selve funktionen, til at tilgå claims for den aktuelle bruger som laver forespørgslen. Herefter verificeres det om denne brugers Username passer med det Username som hører til det Save objekt der hentes.

8.4.2 UserController

UserController klassen skal modsat SaveController klassen tillade anonyme kald, derfor tilføjes attributten AllowAnonymous. På Figure 25 ses Login funktionen.

```
[HttpPost("Login"), AllowAnonymous]
0 references | magnusblaa, 1 day ago | 2 authors, 2 changes
public async Task<ActionResult<Token>> Login(UserDTO userDTO)

{
    userDTO.Username = userDTO.Username.ToLower();
    var user = await _context.Users.Where(u => u.Username == userDTO.Username).FirstOrDefaultAsync();
    if (user != null)
    {
        var isValid = BCrypt.Net.BCrypt.Verify(userDTO.Password, user.Password);
        if (isValid == true)
        {
            return new Token {JWT = GenerateToken(userDTO)};
        }

    }
    ModelState.AddModelError(string.Empty, "Wrong Username or Password");
    return BadRequest(ModelState);
}
```

Figure 25: Code snippet af HttpPost metode som modtager et UserDTO object og retunerer en JWT hvis brugeren findes og password'et er korrekt

Hvis brugeren findes og pasword'et er korrekt retuneres en ny JWT, og brugeren er logget ind.

Til at Hashe en bruger passwords med anvendes Bcrypt [**Bcrypt**]. Bcrypt indeholder funktionen HashPassword("password", BcryptWorkfactor), denne funktion skal have en BcryptWorkfactor, samt et password. Funktionen verify("password", hashedpassword) bruges til at verificere om et givent password svarer til dets krypterede udgave.

8.4.3 BackEndController Client

BackEndControlleren benytter sig af et HttpClient objekt til at håndtere http request/response, og et Token objekt til at holde på den modtagne JWT. Et klasse diagram over BackEndControlleren kan ses på Figure 26

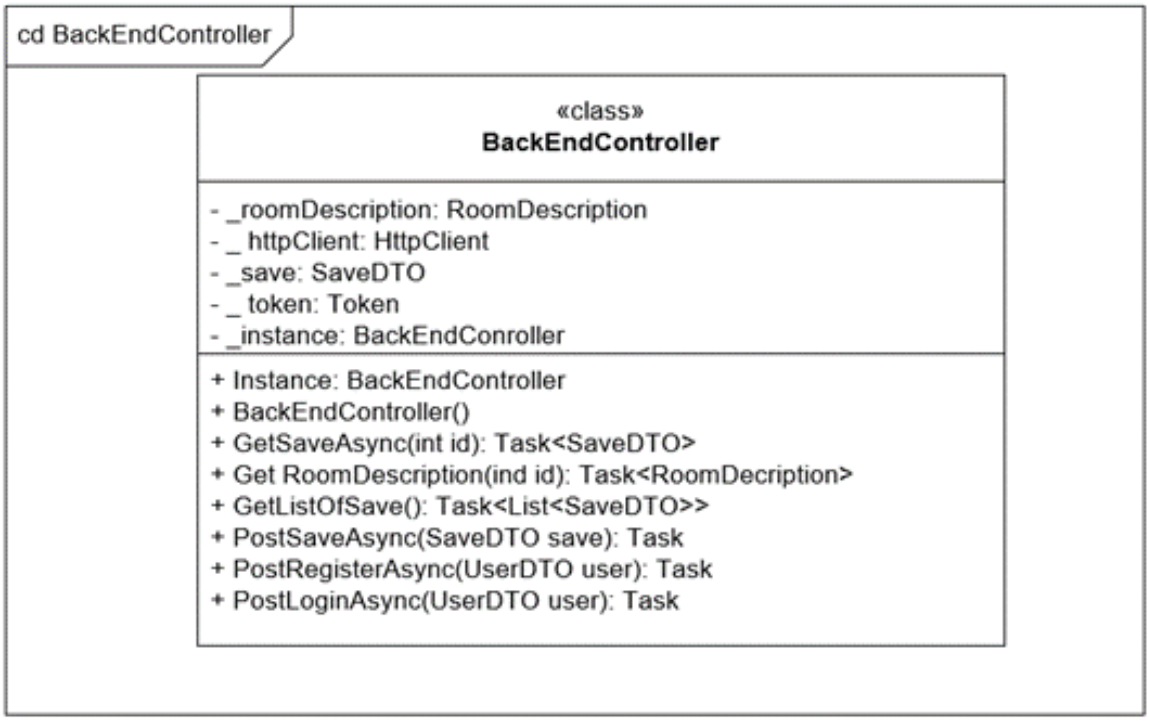

Figure 26: klasse diagram af BackEndController

BackEndController klassen implementeres som en singleton. Dette gøres for at sikre at der kun findes en JWT i programmet af gangen og for at BackEndControlleren instansen altid indeholder den korrekte JWT uanset hvorhenne i client applikationen der laves en request fra.

8.4.4 Konlusion

Backend'ens Funktioner implementeres som asynkrone funktioner, der først retunerer en værdi når resourcen som efterspørges er klar. Client klassen BackEndController implementeres som en singleton for at holde styr på den aktuelle JWT.

8.5 Database Implementering

Til implementering og håndtering af databasen i .NET har gruppen valgt at benytte Entity framework core. EF core gør det nemt at oprette og håndtere objekter C# som skal gemmes eller hentes i databasen.
Til modellering af databasen benyttes det udarbejdede ER-diagram hvorpå keys og relations er bestemt.
Disse keys og relations opsættes i projektets backend ved hjælp af fluent api, hvori man nemt kan specificere keys, foreign keys, relations, hasData og meget mere.
Ved ændringer af databasens udseende og form kan EFcore migrations hjælpe med at oprette de korrekte queries således at databasen bliver ændret korrekt, samt at man ved forkerte ændringer hurtigt kan hoppe tilbage til en tidligere migration ved eventuelle fejl. I EF core benyttes Language Integrated Query (LINQ) til at skrive ensartede queries til databasen.
Der er som gruppe taget en beslutning om at hoste databasen lokalt i en docker container. Dette blev valgt på baggrund af en samtale med vejleder, på baggrund af, sikkerhedsforanstaltning på au's netværk og problemer med microsoft licenser.

Der er i projektets backend oprettet klasser, models, tilsvarende ER diagrammernes entiteter. Udover det valgte atributter er der også oprettet navigationals i de nødvendige klasse så der kan laves relationer. På Figure 27 herunder ses opsætningen af keys for de forskellige entiteter. Disse er opsat med fluent api efter ER diagrammerne på Figure 21 og Figure 22.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    /************ KEYS ************/
    //User
    modelBuilder.Entity<User>()
        .HasKey(x => x.Username);

    //Save
    modelBuilder.Entity<Save>()
        .HasKey(x => x.ID);

    modelBuilder.Entity<Save>()
        .HasIndex(n => new { n.SaveName, n.Username }).IsUnique();

    //RoomDescriptions
    modelBuilder.Entity<RoomDescription>()
        .HasKey(x => x.RoomDescriptionID);

    //Rooms
    modelBuilder.Entity<VisitedRooms>()
        .HasKey(x => new { x.SaveId, x.VistedRoomId});

    //Puzzles
    modelBuilder.Entity<Puzzles>()
        .HasKey(i => new { i.Save_ID, i.Puzzles_ID, });
    //Inventory
    modelBuilder.Entity<Inventory_Items>()
         .HasKey(k => new { k.SaveID, k.ItemID });
    //Enemies
    modelBuilder.Entity<Enemies_killed>()
         .HasKey(k => new { k.SaveID, k.EnemyID });
```

Figure 27: Opsætning af keys og unikke indexes for de forskellige entiteter i backends DbContext

Udover de forskellige keys, skal der også opsættes relationer mellem de forskellige entiteter, som vist på ER-Diagrammerne Figure 21 og Figure 22, samt referencer til foreign keys. Opsætningen kan ses på Figure 28 herunder:

```
// one to many relationship save vistedRooms
modelBuilder.Entity<VisitedRooms>()
    .HasOne<Save>(x => x.Save)
    .WithMany(y => y.VisitedRooms)
    .HasForeignKey(x => x.SaveId);

//many to one
modelBuilder.Entity<Save>()
    .HasOne<User>(s => s.User)
    .WithMany(s => s.Saves)
    .HasForeignKey(i => i.Username);

//1 save to many Puzzles
modelBuilder.Entity<Puzzles>()
    .HasOne<Save>(i => i.save)
    .WithMany(i => i.Save_Puzzles)
    .HasForeignKey(s => s.Save_ID);

//1 Save to many Items
modelBuilder.Entity<Inventory_Items>()
    .HasOne<Save>(i => i.save)
    .WithMany(s => s.Save_Inventory_Items)
    .HasForeignKey(s => s.SaveID);

//1 Save to many Enemies
modelBuilder.Entity<Enemies_killed>()
    .HasOne<Save>(s => s.save)
    .WithMany(s => s.Save_Enemies_killed)
    .HasForeignKey(i => i.SaveID);
```

Figure 28: Opsætning af relations og foreign keys for de forskellige entiteter i backends DbContext

Databasen er seedet ved hjælp af hasdata funktionen. Her oprettes en enkelt bruger, "Gamer1", med password "123", som hashes ind i databasen. "Gamer1" får derudover også 5 tilhørende "tomme" saves og til slut er der også indsat rumbeskrivelser for hver af de 20 rum. Dette ses på Figure 29 herunder.

```
modelBuilder.Entity<Save>()
    .HasData(
    new Save { RoomID = 0, ID = 2, SaveName = "NewGame2", Username = "gamer1", Health = 10 },
    new Save { RoomID = 0, ID = 1, SaveName = "NewGame1", Username = "gamer1", Health = 10 },
    new Save { RoomID = 0, ID = 3, SaveName = "NewGame3", Username = "gamer1", Health = 10 },
    new Save { RoomID = 0, ID = 4, SaveName = "NewGame4", Username = "gamer1", Health = 10 },
    new Save { RoomID = 0, ID = 5, SaveName = "NewGame5", Username = "gamer1", Health = 10 }
    );

modelBuilder.Entity<User>()
    .HasData(
    new User { Username = "Gamer1", Password = BCrypt.Net.BCrypt.HashPassword("123", 11) });


modelBuilder.Entity<RoomDescription>()
    .HasData(
    new RoomDescription { RoomDescriptionID = 1, Description = "1. The king has died of a magical curse c
    new RoomDescription { RoomDescriptionID = 2, Description = "2. You hear a strange rumble. What is hap
    new RoomDescription { RoomDescriptionID = 3, Description = "3. That was a strange encounter, but the
    new RoomDescription { RoomDescriptionID = 4, Description = "4. The door revealed a tunnel that led to
    new RoomDescription { RoomDescriptionID = 5, Description = "5. This room seems empty and dark. There
    new RoomDescription { RoomDescriptionID = 6, Description = "6. The whelps corpse has a weird smell to
    new RoomDescription { RoomDescriptionID = 7, Description = "7. What is this? It looks like a cellar t
    new RoomDescription { RoomDescriptionID = 8, Description = "8. This badly-lit corridor leads to a rus
    new RoomDescription { RoomDescriptionID = 9, Description = "9. The room is empty, but leads to anothe
    new RoomDescription { RoomDescriptionID = 10, Description = "10. It seems you've stumpled upon a dini
    new RoomDescription { RoomDescriptionID = 11, Description = "11. Wait, a well? A rope is attached to
    new RoomDescription { RoomDescriptionID = 12, Description = "12. The room is lit by a mysterious item
    new RoomDescription { RoomDescriptionID = 13, Description = "13. The dead naked Gnoblin looks horrend
    new RoomDescription { RoomDescriptionID = 14, Description = "14. Ew. The room is filled by a stench o
    new RoomDescription { RoomDescriptionID = 15, Description = "15. The door locks as you enter the room
    new RoomDescription { RoomDescriptionID = 16, Description = "16. The corridor is split between two wa
    new RoomDescription { RoomDescriptionID = 17, Description = "17. The brute-gnoblin put up a good figh
    new RoomDescription { RoomDescriptionID = 18, Description = "18. A door is present and you need a key
    new RoomDescription { RoomDescriptionID = 19, Description = "19. The Gnoblin king has been slayed. I
    new RoomDescription { RoomDescriptionID = 20, Description = "DB says: This is room 20" }
    );
```

Figure 29: Seeding af databasen med en enkel bruger, "Gamer1", 5 tilhørende "tomme" saves og beskrivelser af historien til de 20 forskellige rum

9 Test

Testing er en grundsten i ethvert succesfuldt softwaresystem. Uden testing kan der ikke stilles garanti for et systems opførsel. Nedenstående afsnit dækker alle test foretaget af "Dungeons and Gnoblins" spillet, for at sikre den korrekte opførsel i henhold til kravspecifikationerne.
Her dækkes test af alle de største komponenter, Frontend, Game Engine, Backend og database. Testene inkluderer automatiseret unittests, integrationstest og accepttest.

9.1 Modultest Frontend

Modultesten af Frontenden er lavet i meget tæt samarbejde med Game Enginen. Dette er valgt sådan, at når Game Engine lavede en ny funktion eller funktionalitet, gik Frontend holdet igang med at implementere en visuel repræsentation af denne nye funktion/funktionalitet. Således var det ikke kun en visuel test af at views så godt ud eller at man kunne trykke på en knap, men derimod kunne både frontenden og Game Enginen testes i samarbejde, hvor den reelle funktionalitet for systemet blev testet.

9.1.1 Test metoder

Hver gang der er blevet lavet små eller større ændringer i frontenden, er der blevet lavet både en funktionel og visuel test af de nye ændringer. Dette blev gjort for æstetiske ændringer ved at kigge i preview vinduet i WPF for det view der blev ændret. Var det derimod en ændring der inkluderede

databinding til Game Enginen, blev det nødvendigt at teste hele programmet ved brug af compileren og derved lave en runtest, som inkluderede at det rigtige data blev hentet fra Game Enginen eller at den rigtige funktionalitet blev kaldt ved et tryk på en knap.
For eksempler på hvordan Frontenden specifikt er blevet testet i forbindelse med Game Enginen se Teknisk Bilag **(INDSÆT REFERENCE HER)**.

9.2 Game Engine Modul Test

Nedestående præsenteres en delvis tabel for alle klasser testet som led i Game Engine. De vigtigste er GameController, CombatController og DiceRoller. Dice danner "Core Mechanics" for spillet.
Interessant nok ses her at GameController fejler sine test grundet at der ikke er skrevet test til mange af GameControllerens ansvars punkter.

Table 1: Her Ses en komplet liste over alle test foretages på Game Engine komponenter, med kommentar til deres resultater og en endelig vurdering af test resultaterne.

Game Engine GodkendelsesTabel			
Komponent under test	**Forventet Adfærd**	**Kommentar**	**Test Resultat**
Game Controller	1. Kan skifte Player til Nyt Room 2. Kan samle Item op fra Room 3. Kan save Game 4. Kan loaded Games 5. Kan eliminere Enemy fra Game 6. Kan reset Game 7. Kan anskaffe Room description	Game Controller er kun test for at skifte til nyt Room, dette betyder at der ikke kan stilles garanti for at resterende implementeringer af load- og save game osv. fungere som ønsket. Disse ting er svagt testet gennem visuelt trial and error test, men da der ikke er skrevet nogen specifikke test til dem Fejler GameControlleren sin komponent test.	FAIL

Combat Controller	1. Kan Håndtere Combat Rounds 2. Kan håndtere at Player løber væk fra Combat	Combat controller kan håndtere at spilleren løber fra combat og at Player indgår i combat. CombatController kan stille garanti for at combat sker i den rigtige orden og at hverken spiller eller enemy kan angribe hvis denne er død. Ydmere stiller den garanti for at både enemy og spiller kan lave "critial hits" hvis dice rolleren slår 20.	OK
DiceRoller	1. Kan emulerer et kast med en N siddet terning 2. Kan emulerer N kast med en M siddet terning	DiceRoller Kan emulere et eller flere terninge kast med samme antal sidder. Denne kan ydmere stille krav for at fordellingen af disse terningekast har en normal distribution og dermed er alle udfald lige sandsynlige.	OK

9.2.1 Test Resultater for Game Engine

Nedenstående vises kort resultatet for Game Engine test [Section 12.2.1 2], når de alle køres via visual studio. Som det kan ses så er alle testene succesfulde, hvilket giver hvis garanti for at game engine opfører sig som specificeret i kravspecifikationerne og i de implementerede interfaces.

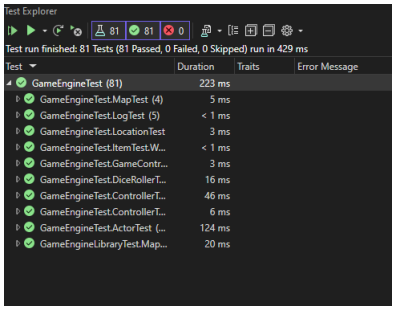

Figure 30: Alle skrevene test til Game Engine passer, hvilket hjælper med at give vished om at Game Engine udfører dens funktionalitet, som det er beskrevet i kravene. Dette siges da alle testene er skrevet på baggrund af kravene som black-box tests og ikke som white-box test efter implementeringen.

Disse resultater giver en god sikkerhed for at game controlleren fungerre som den skal. Desværre er mængden af test mangelfuld og derfor kan der ikke stille nogen garanti for at game engine fungerer hundrede procent som forventet.

9.3 Modultest database

Den lokale hosting medførte at vi kunne holde øje med databasens udformningen, samt den gemte data, gennem Microsoft azure datastudie.

For at teste DAL funktioner og dens forbindelse til databasen, oprettes først et DAL objekt med en context som indeholder en korrekt connectionstring.
Til test hvor der skal indsættes data i databasen oprettes der objekter af den korrekte type hvorefter DAL funktioner til indsættelse kaldes. Korrektheden af de indsatte data kan herefter tjekkes med datastudie. Et eksempel på dette kan ses på Figure 31 herunder. Her oprettes et GameDTO objekt, gamesave.
Gamesave opsættes til "Gamer1", med navnet "My First Run", hvorefter der tilføjes yderligere data. Det overskrevne save er "NewGame1", som her har id 1.

```
var DataHelper = new DAL();

var gamesave = new GameDTO();
gamesave.Armour_ID = 1;
gamesave.Username = "Gamer1";
gamesave.SaveName = "My First Run";
gamesave.RoomID = 13;
gamesave.Health = 69;
gamesave.itemsID.Add(12);
gamesave.enemyID.Add(1);
gamesave.PuzzleID.Add(1);
gamesave.VisitedRooms.Add(1);
gamesave.VisitedRooms.Add(2);

DataHelper.SaveGame(gamesave);
```

Figure 31: Kode til test af SaveGame, hvor der oprettes et nyt save som overskriver det gamle save med saveID 1

På Figure 32 herunder ses et screenshot fra datastudie hvor de 5 saves til "Gamer1" kan ses. Det noteres at alle saves er "tomme" og starter i rum 1.

Results Messages

	ID	RoomID	Armour_ID	Health	Weapon_ID	Username	SaveName
1	1	0	NULL	10	NULL	gamer1	NewGame1
2	2	0	NULL	10	NULL	gamer1	NewGame2
3	3	0	NULL	10	NULL	gamer1	NewGame3
4	4	0	NULL	10	NULL	gamer1	NewGame4
5	5	0	NULL	10	NULL	gamer1	NewGame5

Figure 32: Screenshot fra datastudie før opdatering af save hvor de 5 ens "tomme" saves kan ses

Herefter køres programmet fra Figure 31, og vi kan nu se ændringerne i databasen. Det noteres at SaveName og de andre attributter i Figure 33 nu er opdateret korrekt efter koden.

Results Messages

	ID	RoomID	Armour_ID	Health	Weapon_ID	Username	SaveName
1	1	13	NULL	69	NULL	gamer1	My First Run
2	2	0	NULL	10	NULL	gamer1	NewGame2
3	3	0	NULL	10	NULL	gamer1	NewGame3
4	4	0	NULL	10	NULL	gamer1	NewGame4
5	5	0	NULL	10	NULL	gamer1	NewGame5

Figure 33: Sceenshot af datastudie efter overskrivning af save 1. Her er saveID 1 opdateret til nu at hedde "My First Run" og health og roomID er ændret korrekt, så det passer med det indsatte

De øvrige tilhørende lister til det gemte save er også opdateret med korrekte værdie. Dette kan også ses på Figure 34 herunder.

ItemID	SaveID
13	1

EnemyID	SaveID
2	1

VistedRoomId	SaveId
1	1

Puzzles_ID	Save_ID
3	1

Figure 34: Sceenshot af datastudie efter overskrivning af save 1 med ekstra tabeller som referer til det rigtige saveID

I Table 2 herunder ses en tabel over udførte test, forventede resultater, den faktiske observering og vurderingen af observeringen. Alle test er udført og vi observerer det forventede resultat. Vi vurderer derfor alle test som OK.

Table 2: Tabel over modulttest af DAL forbindelse til databsen. Alle test observeringer stemmeroverens med forventningerne og markeres derfor OK

DAL-Database Godkendelses Tabel				
Test	**Funktion**	**Forventet resultat**	**Observering**	**Vurdering (OK/Fail)**
1	GetRoomDescription(id)	Her forventes det at funktionen henter beskrivelsen fra den valgte rumId.	Funktionen henterbeskrivelsen korrekt og vi kan udskrive denne på konsolen.	OK
2	GetAllSaves	Det forventes at alle spillets saves, med tilhørende info, hentes fra databasen	Alle saves hentes fra databasen og information om disse kan udskrives på konsolen.	OK
3	GetSaveById(id)	Det forventes at der hentes alle oplysninger om et enkelt save fra databasen med tilsvarende id, som den medsendte parameter	Det korrekte save samt tilhørende info hentes og kan udskrive til konsolen	OK
4	SaveGame(Game)	Det forventes at det valgte save med samme id som det nye, overskrives, og at tilhørende info opdateres	Det observeres at det gamle save med samme id, nu er ændret og har korrekte nye værdier	OK

Den ovenstående modultest er generelt godkendet da alle funktioner opfører sig som forventet, i og med at der hentes og gemmes korrekt i databasen. Med alle funktioner testet og godkendt er DAL-database forbindelsen klar til at blive integreret med resten af systemet.

9.4 Backend Modultest

Dette afsnit omhandler modultesten af backenden, denne deles op i følgende to dele, der fortages først en modultest af DAL som kan findes her subsection 9.3 , hvorefter DAL bruges i modultesten af Web api'et, da Web api'et ikke vil blive testet uden DAL modulet.

Til at udføre modultests af web api'et benyttes udviklingsværktøjet Swashbuckle [**Swagger**]. Swashbuckle bruges til at bygge SwaggerDocument objekter på baggrund af de routes, controllers og modeller som er blevet udviklet. Hertil tilbyder Swashbuckle et Swagger User interface, hvor man kan teste sine http funktioner og se om man modtager den forventede response. Derfor er dette et oplagt værktøj at gøre brug af. Et billede af Swagger User interfacet kan ses på Figure 35.

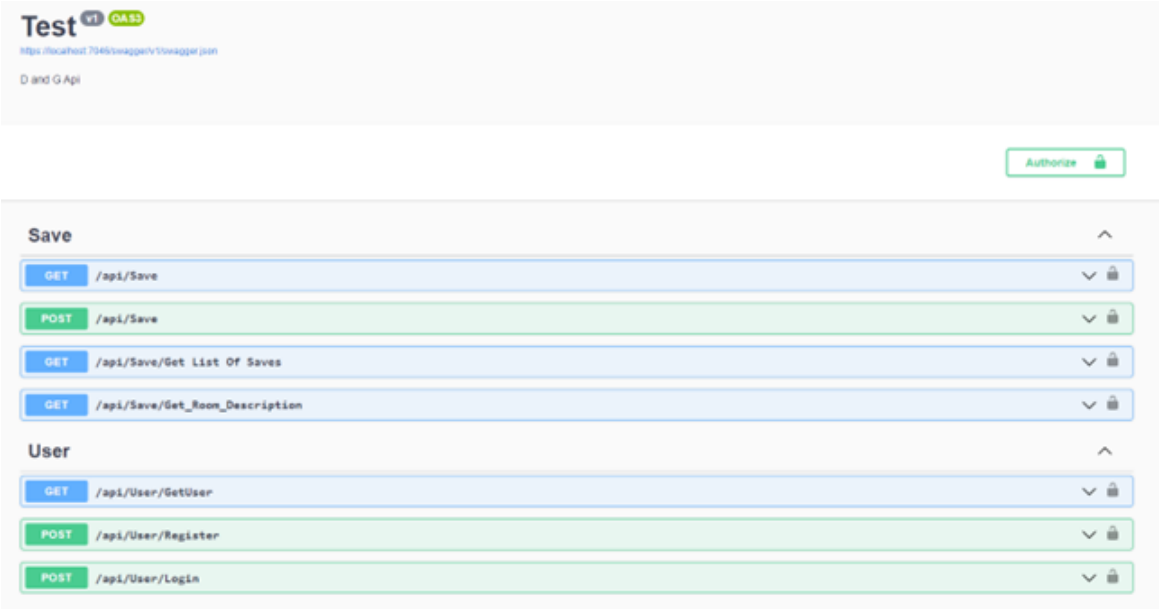

Figure 35: Swagger User interface, som viser de forskellige http metoder

For at aktivere swagger tilføjes følgende middleware til program.cs AddSwaggerGen(), UseSwagger() og UseSwaggerUI(). Til SwaggerGen tilføjes også security med JWT tokens så Authentication og Authorization kan testes.

Der er løbende blevet foretaget modultest, da det er forholdsvist enkelt at benytte swagger. På den måde undgåes det at skulle sidde og rette store mængder af kode på en gang, men i stedet blevet gjort opmærksom på fejl løbende. I Table **??** kan ses en tabel oversigt over tests af succes scenarie, for de enkelte funktioner samt resultater.

Table 3: Tabel over modultest af Backendes Web Api, her vises testes af succes scenarier for alle Web Api http funktioner

Test	**Funktion**	**Forventet resultat**	**Observering**	**Vurdering (OK/Fail)**
1	PostSave(SaveDTO saveDTO)	Der bliver sendt et specifikt gamestate, for brugeren der er logget ind.	Det er muligt at sende game statet, og de korrekte værdier bliver sendt med.	OK
2	Register(UserDTO regUser)	Der registreres en ny bruger, ved at gemme oplysninger omkring denne. Der returneres en JWT-token.	Brugeren bliver registreret, og kan findes blandt de andre brugere. Det ses at der returneres en JWT-token.	OK
3	Login(UserDTO userDTO)	Der tjekkes om oplysningerne passer med en registreret bruger, og passer det logges der ind. Der returneres en JWT-token.	Det lykkedes at logge ind, og der returneres en JWT-token.	OK
4	GetSave(int id)	Der hentes et specifikt save for den bruger der er logget ind.	Det lykkedes at hente et specifikt game state, uden errors	OK
5	GetListOfSave()	Der hentes en liste af game states, for brugeren der er logget ind.	Det lykkedes at hente en liste af game states, for den specifikke bruger.	OK
6	GetRoomDescription(int id)	Denne route henter en beskrivelse af det valgt rum i spillet.	Det ses, at der bliver hentet en beskrivelse af det valgte rum.	OK

I nesdstående tabel testes om funktionerne håndtere fejlscenarier korrekt.

Table 4: Tabel over modultest af Backendes Web Api, her vises testes af fejlscenarier for alle Web Api http funktioner

Test	**Funktion**	**Forventet resultat**	**Observering**	**Vurdering (OK/Fail)**
1	Register(UserDTO regUser)	Brugernavnet er allerede i brug, og der sendes en fejlmeddelelse	Status: 400. Besked: "Name is already in use".	OK
2	Login(UserDTO userDTO)	Oplysningerne stemmer ikke overens med en registreret bruger, og der bør sendes en fejlmeddelelse.	Status: 400. Besked: "Wrong Username or Password"	OK
3	GetSave(int id)	Der er ikke logget ind, så der kan ikke hentes et game state.	Status: 401, Unauthorized	OK
4	GetListOfSave()	Der er ikke logget ind, og derfor kan der ikke hentes et liste.	Status: 401, Unauthorized	OK
5	PostSave(SaveDTO saveDTO)	Der er ikke logget ind, og derfor kan der ikke gemmes er spil.	Status: 401, Unauthorized	OK

Med alle funktioner testet med et godkent resultat, er backenden klar til at blive integreret med de andre moduler for systemet.

10 Integrationstest

10.1 Tidlig Integrationstest

For gruppen var det et mål at komme igang med at lave integrationstest så hurtigt som muligt, da vi ønskede at der hurtigt kom styr på kommunikationen mellem de forskellige moduler i systemet. Her integreres altså blot vores **MVP REFERENCE HER**og det ville derefter være nemmere at implementerer nye features.

For at køre systemet har vi først startet den oprettede docker container til spillets database. Derefter startede vi spillets server, i form af backend api. Til slut kunne spillets klient åbnes og systemet testes.

I den tidlige integrationstest løb vi som gruppe ind nogle forskellige problemer.
Vi havde som gruppe arbejdet for opdelt i forskellige dokumenter. Derudover opdagede vi at de forskellige projekter benyttede forskellige versioner af .net, dette var dog hurtigt løst.
Til slut opdagede vi at der, når man loader et spil, ikke blev vist for brugeren hvilken rum man havde besøgt. Dette blev diskuteret og gruppen besluttede at tilføje dette til næste iterationer.

I **INDSÆT REF TIL VIDEO HER** ses en demovideo af integrationstesten af MVP. Her det ses at man kan spille spillet, gemme det og derefter loade det korrekt ind igen, dog igen uden at man kan se hvilke rum man har besøgt.

10.2 Fulde Integrationstest

Med den indledende integrationstest, er kommunikationen igennem sytemet på plads, samt de basale funktionaliteter. Herfeter udvides spillet med flere features som: Login, Register, Enemies, Items, Health, EquipedItems, Shield, Combat, Map Visibility og Room Descriptions. Dette er features, som kræver yderligere udvikling indenfor alle tekniske områder af projektet.

Feature udviklingen i Frontend og Game Engine Modul udvikles sammen, derfor er de moduler allerede integreret med hinanden, det samme gælder for Backend og Databasen.

Der laves endnu en integrations test med alle de nye features. Her opstod en række nye problemer hvad angår Http kommuniaktionen imellem backend'en og frontend'en. Dette bundede i overenstemmelsen af data modellerne på backend og frontend siden. Det drejede sig helt specifikt omkring de nyeligt tilføjede properties til modellen "Save", på fronted'en blev brugt datatypen uint imens der på backend blev brugt int, samtidigt med at properties hed noget forskelligt. Derfor blev de nye properties ignorede når data'en blev seraliseret og deseraliseret. Dette problem var ikke noget som debuggeren gjorde opmærksom på derfor tog det relativt lang tid at få det løst.

Et andet problem som opstod, var angående Room Description, her blev det besluttet under implementering at gemme dem i databasen fremfor lokalt, grundet problemer med resources og deres specifikke stier på forskellige pc'er.

Med disse problemer løst, resulterede integrationstesten i vores færdige produkt, som blev anvendt i acceptesten. Acceptesten samt en demonstrations video af produktet kan ses her. **REF TIL VIDEO AF ACCEPTEST VIDEO**

11 Accepttestspecifikation

For accepttestspecifikationen er der for dette projekt, opstillet tests, og på baggrund af disse kan det betragtes som et funktionelt og acceptabelt produkt. Denne accepttestspecifikation er delt op i Funktionelle krav og ikke funktionelle krav.

11.1 Funktionelle

Ved de funktionelle krave er der her taget udgangspunkt i User Story 1,2,7 og 17, da disse indebærer alle systemets dele samt den mest centrale funktionalitet.
De resterende accepttest kan findes **REF HER**

Test af User Story 1 - Login 1. Scenarie - Succesfuld login
Givet at brugerens profil er oprettet på databasen og at serveren er oppe.

- Når bruger indtaster sit login(Brugernavn og password)
- og trykker "Log in" på UI
- Så skifter skærmen til hovedmenu

Resultat: Godkendt
Kommentar:

1. Scenarie - Fejlet login
Givet at brugerens profil er oprettet på databasen og at serveren er oppe.

- Når bruger indtaster et forkert login(Brugernavn og password)
- og trykker "Log in" på UI
- Så signaleres der om forkert login-oplysninger til bruger

Resultat: Godkendt
Kommentar: Viser korrekt error besked

Test af User Story 2 - Opret profil 2. Scenarie - Bruger opretter profil
Givet at brugerens profil er oprettet på databasen og at serveren er oppe.

- Når bruger vælger at oprette profil
- Så gemmes data for brugerens profil på databasen
- Og skærmen skiftes til log ind skærmen

Resultat: Godkendt
Kommentar:

2. Scenarie - Bruger opretter samme profil
Givet at brugerens profil er oprettet på databasen og at serveren er oppe.

- Når bruger vælger at oprette en allerede-eksisterende profil
- Så signaleres der om at profil allerede eksisterer

Resultat: Godkendt
Kommentar: Viser korrekt error besked

Test af User Story 15 og 18 15. Scenarie - Save Menu - Save Game
Givet at brugeren har trykket "Save game" fra Settings menu og ikke er i combat.

- Når bruger vælger et gemt spil
- Og sætter navnet til det ønskede
- Og trykker på "Save Game" knappen
- Så genoptages spillet

Resultat: Godkendt
Kommentar:

18. Scenarie - Main menu - Load Game - Load
Givet at brugeren er logget ind og har adgang til spillet og trykket "Load Game"

- Når bruger vælger et gemt spil
- Og trykker "Load Game"
- Så loader spillet det gemte spil med det valgte game state

Resultat: Godkendt
Kommentar: Bruger starter i korrekt rum med korrekt inventory

11.2 Ikke-funktionelle

Herunder er udfyldte accepttest for systemets ikke-funktionelle krav. Her tages der igen et udpluk, som indeholder de mest kritiske krav. De resterende kan findes her**REF HERE DATABASE**

Table 5: Fuldført ikke funktionelle tests for DATABASE

Beskrivelse	Verificering	Resultat	Kommentar
Skal kunne gemme maksimalt 5 save games	Visuel	Fejl	Front-end begrænser antal gemte spil, databasen kan indeholde flere gemte spil end fem.
Skal kunne loade et spil indenfor maksimalt 5s	Visuel	Godkendt	
Skal gemme hvilke genstande man bruger lige nu	Visuel	Godkendt	
Skal gemme hvor meget liv man har tilbage.	Visuel	Godkendt	
Skal gemme hvilke fjender man har slået ihjel.	Visuel	Godkendt	
Skal gemme hvilke puzzles man har løst	Visuel	Fejl	Puzzles er ikke implementeret
Skal gemme hvilke rum man har været i.	Visuel	Godkendt	

GAMEPLAY

Table 6: Fuldført ikke funktionelle tests for GAMEPLAY

Beskrivelse	Verificering	Resultat	Kommentar
Spillets kort skal holde styr på hvilke rum man kan komme til for et givet rum.	Visuel	Godkendt	
Spillets kort skal kun vise de rum som spilleren har været i.	Visuel	Godkendt	
Spillets kort skal, hvis spilleren har været i alle rum vise alle rum.	Visuel	Godkendt	
Et rum kan have maksimalt 4 forbindelser til andre rum.	Visuel	Godkendt	
Et rum skal have mindst 1 forbindelse til andre rum.	Visuel	Godkendt	
Alle Rum skal kunne nås fra ethvert andet rum, måske ikke direkte, men man skal kunne komme dertil.	Visuel	Fejl	Tutorial rum er ikke tilgængeligt, når bruger har forladt det.
Spillerens rygsæk skal kunne indeholde alle spillets genstande.	Visuel	Godkendt	
Spilleren skal have mulighed for at bruge ét våben og ét skjold af gangen.	Visuel	Godkendt	
Spilleren skal have mulighed for at skifte hvilket våben og hvilket skjold der bruges.	Visuel	Godkendt	

COMBAT

Table 7: Fuldført ikke funktionelle tests for COMBAT

Beskrivelse	Verificering	Resultat	Kommentar
Når spilleren/fjenden prøver at slå, rammer man kun hvis man på sit angreb slår højere end modstanderens rustningsværdi. Dette afgøres af et simuleret 20 sidet terninge kast, hvortil der lægges en værdi til, korresponderende til spilleren/fjendens våben bonusser.	Visuel	Godkendt	Et angreb går også igennem hvis slaget er lige på PC's rustningsværdi
Hvis spilleren/fjenden rammer, bliver skaden bestemt af et/flere simulerede terningekast, afhængigt af hvilket våben der bruges.	Visuel	Godkendt	
Hvis spilleren/fjenden rammer, bliver skaden bestemt af et eller flere simulerede terningekast, afhængigt af hvilket våben der bruges.	Visuel	Godkendt	
Hvis spilleren, når nul liv inden fjenden, så dør spilleren og spillet er tabt.	Visuel	Godkendt	
Hvis fjenden, når nul liv inden spilleren, så dør fjenden og spilleren kan nu frit udforske rummet, som fjenden var i.	Visuel	Godkendt	
Hvis spilleren drikker en livseleksir bliver spillerens nuværende liv sat til fuldt	Visuel	Fejl	Livseliksir er ikke implementeret

PERFORMANCE

Table 8: Fuldført ikke funktionelle tests for PERFORMANCE

Beskrivelse	Verificering	Resultat	Kommentar
Spillet skal respondere indenfor maksimalt 5s	Visuel	Godkendt	
Spillet må ikke have mere end én kommando i aktionskøen af gangen	Visuel	Fejl	Aktions kø er ikke blevet implementeret

11.3 Diskussion af testresultater

Projektets implementering og kravspecifikationerne har produceret adskillige tests, som projektet i overvejende grad har bestået. De fleste features er blevet testet i henhold til kravspecifikationerne se section 3 og har produceret resultater, der indikerer at hver feature fungerer som ønsket.

Relevante funktionelle tests er beskrevet med user-stories og er evalueret med et "Godkendt" eller "Fejl" Table 1 og evt. en foklarende kommentar. De resterende udfylte test kan findes i **REFERENCE TIL ACCEPTTEST BILAG** Ikke funktionelle test har fået en evaluering "OK" eller "FEJL". Langt størstedelen af alle tests er bestået med få tests som fejler. Nogle tests fejler, da implementeringen ikke blev som forventet, mens andre ikke er blevet implementeret på grund af tidspres.

Et eksempel på dette er "puzzles" som skulle have været implementeret, som en del af det færdige spil. Grundet tidspres er denne features ikke blevet implementeret og nedprioteret til fordel for andre features, såsom "Combat" der er blevet vurderet mere essentiel.

Et andet eksempel er"Delete Save Game" som ikke blevet implementeret som specificeret i kravspecifikationerne men eksisterer som evnen til at overskrive allerede eksisterende save games.
Den stores success rate i test skyldes en insistens på tidlige integration mellem alle store komponenter for at sikre, at alt kommunikationn har fungeret fra et tidligt tidspunkt i projektet. Det har vist sig nemmere at integrerer mange små ændringer i de individuelle moduler, end at lave en stor integration til sidst i udviklingsfasen.

Lad der dog ikke herske tvivl om, at der er store huller i projektets test suit, det betyder at store features ikke er testet i tilstrækkeligt omfang. Features som load- og save game er implementeret og testet ved visuel bekræftelse, men der er en betydelig mangel på modultest til yderligere at bekræfte, at disse feature fungerer som ønsket (se section 3).

12 Fremtidigt Arbejde

Sammenlignes det færdige produkt med det produkt der blev diskuteret under idefasen, er der nogle funktionaliteter som ikke er blevet færdiggjort, men kunne have været implementeret. Spillet kunne have haft flere udvidelser såsom Puzzles, flere niveauer, character udvikling og quest items som kunne have givet bruger en mere underholdende oplevelse og mere avanceret gameplay. Dette høre ikke under et specifikt modul i systemet, men tilføjes over alle moduler, hvis nogen af de overstående

features blev tilføjet.

F.eks. hvis der blev tilføjet flere niveauer, skulle der implementeres et nyt map i Game Controller og Front-end siden. Derudover skal der tilføjes en udvidelse af load og save game, så Back-end og database kunne gemme specifikt for det nye niveau, hvilket niveau er spilleren på, hvor langt brugeren er på det nuværende niveau og til sidst hvor meget spilleren har udforsket af tidligere niveauer.

Specifikt for Database og Back-End, kunne der tilføjes en lagring af data på en cloud-based storage fremfor lokal storage. Der blev valgt under forløbet at lagre dataen i en local storage, da der i midten af semestret var problemer med skolens licens af Microsoft. For ikke at komme ud for udfordringer senere i forløbet, blev der valgt at gå med den sikre løsning, at hoste databasen lokalt på enheden. Resultatet af et cloud-based lagring ville resultere i at spillet ville være mere tilgængelig for brugere på forskellige enheder eller platforme.

Derudover kunne spillet laves til at kunne køre på andre styresystemer som f.eks. IOS eller Unix. Dette ville lade en bredere målgruppe spille spillet og derved give en større kundegruppe. Denne ændring ville dog resulterer i at det valgte framework til Frontenden skulle skiftes til noget andet, såsom Unity. Dette ville betyde at frontenden og backenden skulle omskrives helt, men det kunne også give en generel bedre spil oplevelse.

13 Konklusion

Gruppen er endt ud med et funktionelt projekt, hvor de vigtigste features, blev implementeret. Disse inkludere succesfuld load og save games, rum spilleren kan bevæge sig i, kamp mellem spilleren og fjender, og en frontend der gør spilleren istand til at inteagere med spillet.
Nogle features er ikke blevet implementeret, og er derfor sat til fremtidigt arbejde. Nogle af disse features inkludere gåder og puzzels samt en evne til at slette save games. Disse features er ikke anset som værende nødvendige for at opnå et funktionelt spil, som kunne spilles igennem og stadig fungere som et spil med fjender og genstande, som kunne samles op og anvendes.

Fokus på kommunikation mellem systemets segmenter og integrationen af disse segmenter har været et kerne mål gennem hele projektet og har leveret et spil hvor alle modulerne på succesfuld vis kan kommunikere med hinanden.
Modulerne er lavet med fremtidigt arbejde i tankerne, så f.eks. i Game Module er det lavet relativt nemt at implementere yderligere features, så fremtidige features kunne implementeres uden at komprimere kommunikation mellem systemets segmenter eller med svage modifikationer dertil.
Modulerne er gennem testet og fungere som forventet, dog ikke til den fulde standard specificeret i continuous integration, men således at der god vished om at de fungere som forventet. Skulle projektet udvides burde disse test omdannes til fuldt continuous integration for at sikre at nye features inteageres på bedst mulig vis.

I sidste ende er slut produktet et glimerende eksempel på en tidlig prototype til et spil, der, givet mere tid, kan blive et meget underholdene text-based adventure game.

References

[1] Andy. *Navigating between views in WPF / MVVM*. Visited 12/04-2022. URL: `https://www.technical-recipes.com/2018/navigating-between-views-in-wpf-mvvm/`.

[2] Magnus Blaabjerg Møller et al. *Teknisk Bilag - Dungeons and Gnoblins*. 2022.

14 Bilag

Til raporten medfølger fire bilags dokumenter: Et teknisk bilag (.pdf), et process bilag (.pdf), source code (folder) og en accepttest video (.mkv). Hver af de fire dokumenter indeholder en række bilag:

- Teknisk bilag
  - Krav specifikation og Acceptest
  - Analyser
  - Arkitektur, design, implementerin og test af software moduler
  - Integration af systemet
- Proces bilag
  - Procesbeskrivelse
  - Samarbejds kontrakt
  - Tidsplan
  - Møde indkaldelser og referater
- Source code
  - Source code for spillet i "FrontEnd GameLayout" mappen
  - Source code for Back-end og database i "BackEnd"
- Acceptest Video
  - Demonstaration af produktet