
Gnomes and Gnoblins

Semesterprojekt

Aarhus Institute of Technology

Author:

Morten Høgsberg, 201704542

Date: May 25, 2022

Contents

1	Introduction	4
1.1	Spilbeskrivelse	5
1.2	Login-screen	5
1.3	Core gameplay layout	5
1.4	Settings	6
2	Systembeskrivelse	6
2.1	Kravspecifikation	7
2.2	Funktionelle krav - User Stories	7
2.3	Ikke-funktionelle krav	11
2.4	MOSCOW krav	12
2.5	Accepttest	13
3	Arkitektur	18
3.1	Frontend Arkitektur	19
3.1.1	Pseudo Frontend Arkitektur	19
3.2	Database Arkitektur	22
3.3	DAL Arkitektur	23
4	Design	25
4.1	Overordnet System Design	25
4.2	Frontend Design	25
4.2.1	Room	26
4.2.2	Combat	27
4.2.3	Login	27
4.2.4	Settings	28
4.3	Database Design	29
4.4	Frontend Implementering	29
4.4.1	Login view	30
4.4.2	Room View	30
4.4.3	Combat View	30
4.4.4	Settings view	30
4.4.5	Load view	31
4.4.6	Note om Baggrundsfarver	31
5	Implementering	32
5.1	System Implementering	32
5.2	Game Engine	32
5.2.1	Gamecontroller Implementering	32
5.2.2	Generering af Map	33
5.2.3	Log	33
5.3	Kommentar til implementeringen	34
6	Test	34
6.1	Modultest Frontend	34
6.1.1	Test metoder	34
6.1.2	Eksempel på frontend test i forbindelse med Game Engine	34
6.1.3	Eksempel på frontend test	35
6.2	Modultest Game Engine	35
6.2.1	GodkendelsesTabel Game Engine	36

6.2.2	Mock testing	41
6.2.3	Test Resultater for Game Engine	42
6.3	Discussion af Test Result	43

Introduction

Spilbeskrivelse

PC = Player Character RNG = Random Number Generator

Spillet designes som et text-based spil, det er en spilgenre hvor brugeren interagerer med spillet igennem tekst beskrivelser. Spillet består overordnet af fire dele med hver deres ansvar, en grafisk brugergrænseflade, en database, en back-end til netværkskommunikation samt selve spil logikken. Brugergrænsefladen gør det muligt for brugeren at integrere med spillet. Den vil bestå af en række forskellige vinduer med hver sit formål (login screen, gameplay screen og settings screen), der hovedsageligt vil indeholde knapper og beskrivende tekst. Databasens ansvar er at gemme de enkelte spil, det er en relationel database som sammenholder de enkelte brugeres profiler (Username og password) med informationer omkring deres fremskridt i spillet, såsom placering, items og stats. Det skal være muligt for en bruger at hente sine gemte spil ned på flere forskellige enheder, til dette skal der bruges en netværksforbindelse til databasen.

Login-screen

Det første brugeren bliver mødt af en login-screen. Herfra skal brugeren enten indtaste sine login-oplysninger eller oprette sig en profil i systemet. Systemet har en database hvor alle profiler er lagret. Herunder ses en illustration af spillets login-screen.



Figure 1

Core gameplay layout

PC kommer ind i et rum (se billede 2). Brugeren bliver præsenteret med en beskrivelse af rummet, en liste af elementer i rummet, og en række muligheder for at interagere med rummet (interaktionen foregår ved at trykke på knapperne markeret “Button” i billede 2). Når brugeren er færdig med at interagere med rummet, forlader de det ved at vælge en retning (“go north/south/east/west”). Dette fører dem ind i et nyt rum, mappet opdateres og loopet starter forfra. Bevægelsen i spillet kan gøres i de forklarede fire retninger. Disse retninger indikerer for spilleren hvilke rum de kan bevæge

sig ind i relativt til det rum de er i, der kunne f.eks. være rum, som kun har en vej ind og en vej ud, så kan man ikke gå andre veje end vejen man kom ind. North/south/east og west retningerne er baseret på et kompas, så derfor ville North resultere i at bevæge spillerens karakter opad, South nedad osv. Det tilhørende map image.....(diskuteret) Rum på mappet loades efethånden som man besøger dem.

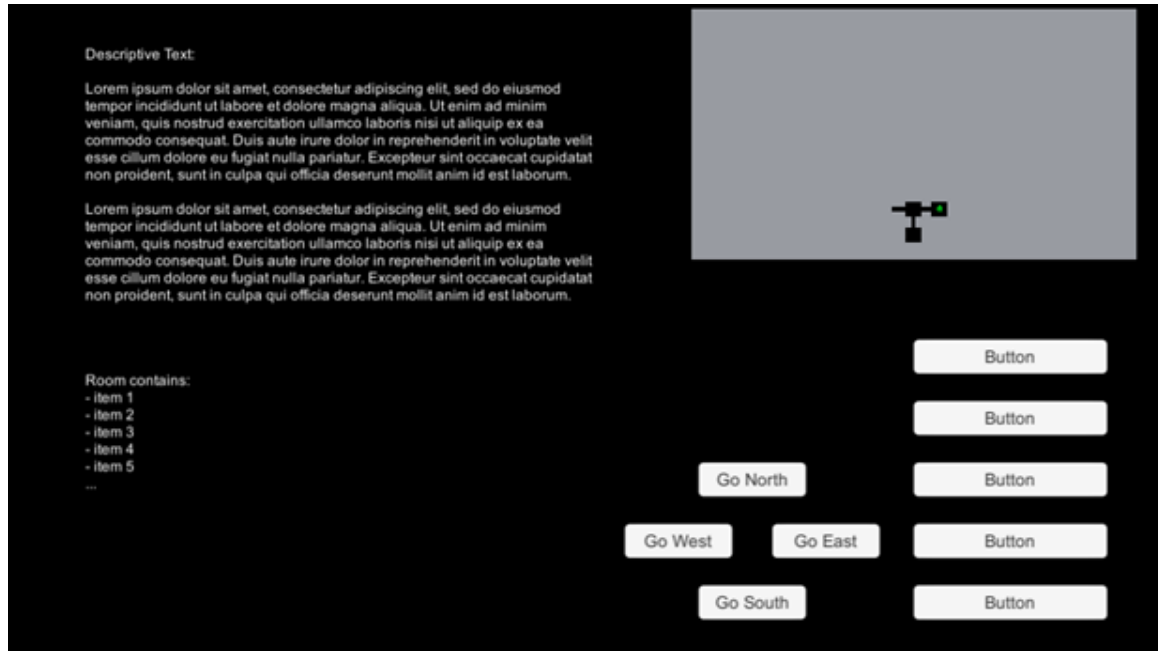


Figure 2

Et rum kan indeholde fjender, som skal bekæmpes før man kan tilgå resten af rummet. Kamp foregår ved at spillet “slår en terning” (RNG) for både PC og fjenden og lægger deres “combat stat” til slaget. Den som slår højest, giver en vis mængde skade til modstanderen, afhængigt af deres “damage stat”. Denne skade trækkes fra karakterens liv. Når en karakters liv bliver mindre eller lig 0 dør karakteren. Hvis spiller karakteren dør taber brugeren spillet. Hvis hverken PC eller fjenden er død efter en kamp får brugeren valget mellem at flygte (gå tilbage til det rum de kom fra) eller fortsætte kampen (start loopet forfra). Det er muligt at finde våben og udrustning i banen, som kan bruges til at forbedre karakterens stats.

Settings

Det skal være muligt for spilleren at tilgå en menu med spillets indstillinger. Her skal det være muligt for brugeren at tilpasse spillets layout indstillinger så som resolution og window size. Det skal ligeledes være muligt at justere lydstyrken for spillet. En illustration af hvordan denne menu kunne se ud er vist på figur x.

Systembeskrivelse

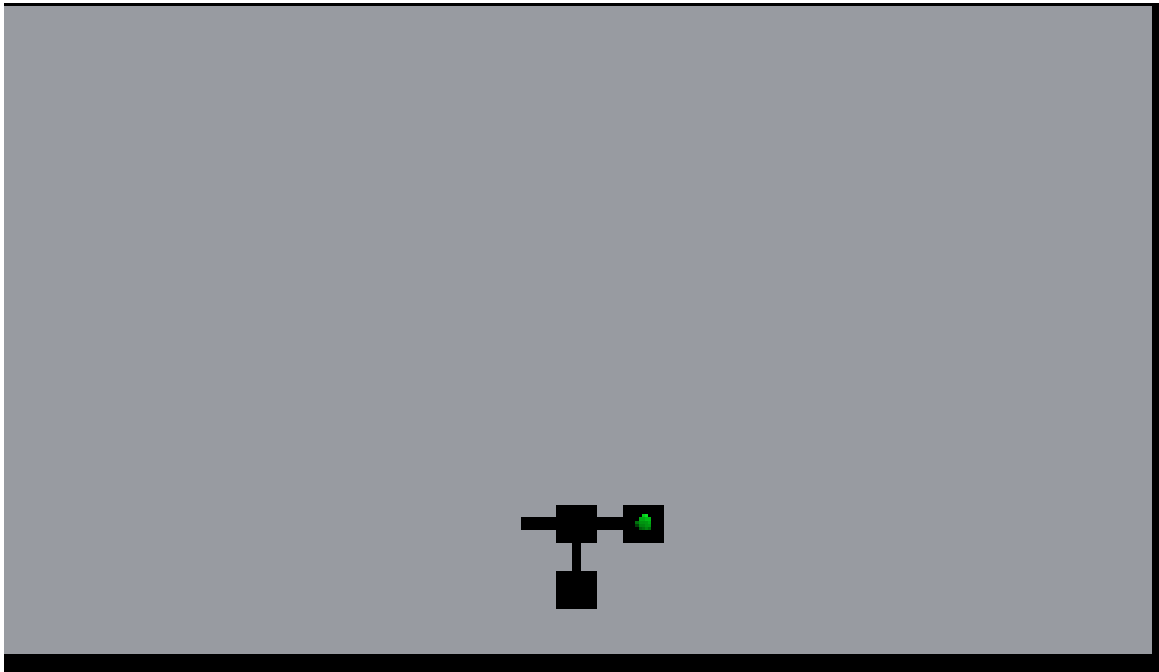


Figure 3: Closeup Screenshot af mappet

Kravspecifikation

I det følgende afsnit vil vi uddybe de forudbestemt krav til projektets specifikation. Dette indeholder en systembeskrivelse, samt en accepttestspecifikation til de opsatte krav. Kravene til systemet vil være opdelt i funktionelle og ikke funktionelle, hvor de opsatte ikke-funktionelle krav vil være prioriteret efter MoSCoW princippet. Der er ydermere opsat accepttest som tester alle scenarier i de opsatte user stories, samt systemets ikke-funktionelle krav.

Funktionelle krav - User Stories

I dette afsnit forklares systemets funktionelle krav i form af user stories som tager os igennem systemet og de ønskede funktioner.

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.

User Story 3: Main Menu - Settings

Som bruger

Kan jeg tilgå Settings

For at jeg kan customize interfacet

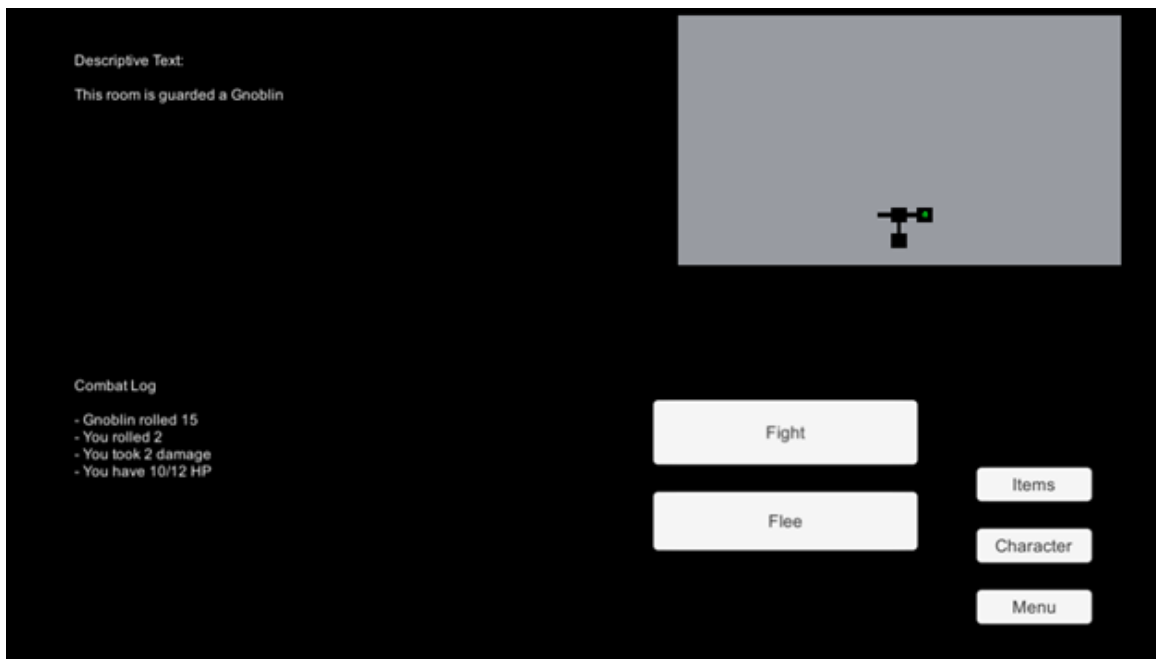


Figure 4

User Story 4: Main Menu – Start Spil

Som bruger

Kan jeg starte et nyt spil

Ved at give spillet et navn som bruges til save games

User Story 5: Exit game

Som Bruger

Kan jeg trykke Esc

For at se en menu med mulighederne "Return to game", "Save and Exit", "Exit without saving" and "Settings".

User Story 6: Exit Menu - Resume Game

Som Bruger

Kan jeg trykke på "Resume game"

For at vende tilbage til spillet

User Story 7: Exit Menu - Save and Exit

Som Bruger

Kan jeg trykke på "Save and Exit"

For at spillet gemmes og bruger bliver vist Main Menu

User Story 8: Exit Menu - Exit Without Saving

Som Bruger

Kan jeg trykke på "Exit Without Saving"

For at få en "Are you Sure" alert

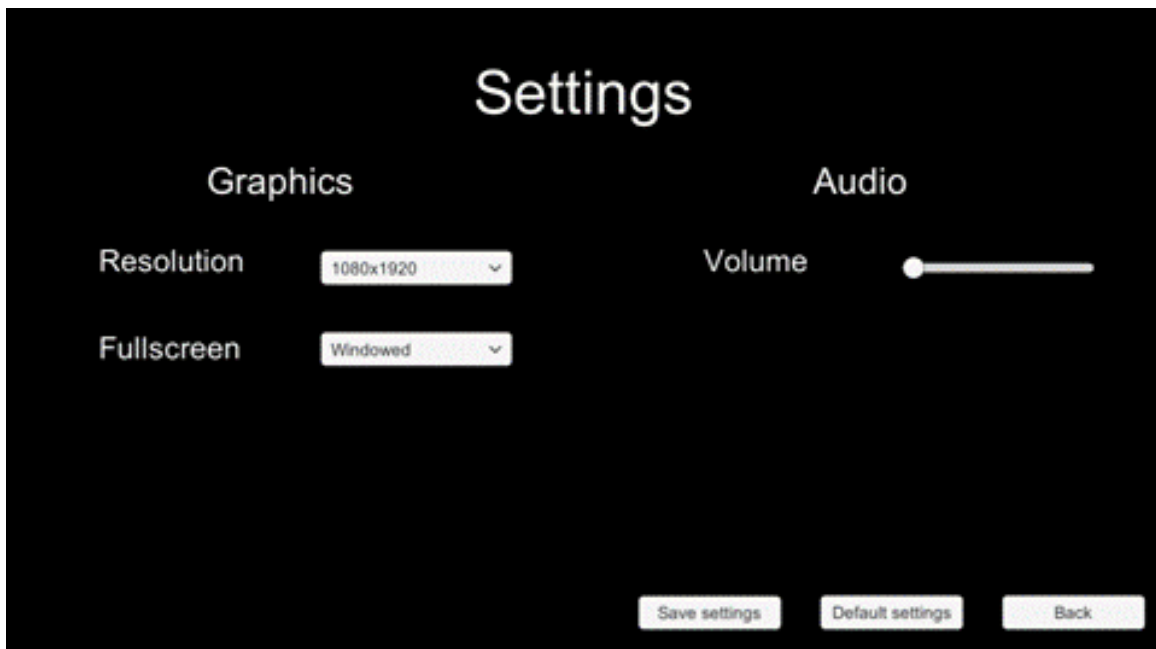


Figure 5: Screenshot af settings menuen

User Story 9: Exit Menu - Exit Without Saving - No
Som Bruger
Kan jeg trykke på "No"
For at vende tilbage til exit menu

User Story 10: Exit Menu - Exit Without Saving - yes
Som Bruger
Kan jeg trykke på "Yes"
For at vende tilbage til Hovedmenuen uden at gemme

UserStory 11 : Exit Menu - Settings
Som Bruger
Kan jeg trykke på "Settings"
For at se Settings Menuen.

UserStory 12 : Settings Menu - Efter ændringer
Som bruger
Kan jeg trykke "Apply", "back", "Cancel" og "Default"
For at tilpasse spillets indstillinger

UserStory 13 : Settings Menu - Efter ændringer - Apply
Som Bruger
Kan jeg trykke "Apply"
For at ændre settings

UserStory 14 : Settings Menu - Efter ændringer - back
Som Bruger

Kan jeg trykke "back"
For at brugeren sendes tilbage til Esc Menu

UserStory 15 : Settings Menu - Efter ændringer - Cancel
Som Bruger
Kan jeg trykke på "Cancel"
For at ændringerne bliver reverted

UserStory 16 : Settings Menu - Efter ændringer - Default
Som bruger
Kan jeg trykke på "Default"
For at sætte settings til defaults

UserStory 17 : Main Menu - Load Game
Som bruger
Kan jeg trykke på "load game"
For at få en liste a save games.

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

UserStory 19 : Main Menu - Load Game - Delete Game
Som bruger
Kan jeg trykke "delete game"
For at game statet bliver slettet fra cloud.

UserStory 20 : Main Menu - Load Game - Back
Som bruger
Kan jeg trykke "back"
For at vende tilbage til Main Menu

UserStory 21 : Spil Spillet - Bevægelse i spil
Som bruger
Kan jeg vælge mellem actioner ved at bruge tallene 0-9 eller piltasterne til at bevæge mig
For at bevæge karakteren i spillet til et andet rum.

UserStory 22 : Spil spillet - Enter nyt room
Som bruger
Får jeg en beskrivelse af et rum, når jeg kommer ind i rummet
Så jeg ved hvad jeg kan interagere med.

User Story 23: Spil spillet - Inventory
Som bruger
Kan jeg trykke på "Inventory" knappen
For at se genstande som jeg besidder

UserStory 24: Spil spillet - Combat
Som bruger
Kan jeg trykke på "attack" knappen

For at skade fjenden

User Story 25: Spil spillet - Combat - flygt
Som bruger
Kan jeg trykke på "flee" knappen
For at blive rykket tilbage til forrige rum

UserStory 26: Spil spillet - Inventory
Som bruger
Kan jeg trykke på "Inventory" knappen
For at tilgå karakterens udrustning

UserStory 27: Spil spillet - Interact
Som bruger
Kan jeg trykke på "interact" knappen
For at interagere med objekter i det nuværende rum

UserStory 28: Spil Spillet - Level klaret
Som bruger
Kan jeg færdiggøre det sidste rum
For at få en besked om at spillet er fuldført og mulighed for at tilgå Main Menu

Ikke-funktionelle krav

I dette afsnit opsættes systemets ikke-funktionelle krav. Disse er opdelt efter kategori, hvorved der er krav til FURPS modellens forskellige dele. Ikke-funktionelle:

GUI:

- GUI'en skal tilbyde valget mellem 3 resolutions
- GUI'en skal kunne være fullscreen
- GUI'en skal kunne være windowed

SOUND:

- Lyden skal kunne justeres mellem 0-100% relativt til PC'ens lyd niveau.

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:

- Spilleets kort skal holde styr på hvilke rum man kan komme til for et givet rum.
- Spilleets kort skal kun vise de rum som spilleren har været i.
- Spilleets 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.
- 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.
- Hvis spilleren/fjenden rammer, bliver skaden bestemt af et/flere simulerede terningekast, afhængigt af hvilket våben der bruges.

PERFORMANCE:

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

STABILITY:

- MEANTIME BETWEEN FAILURE - 1Time +/- 10Min

DOCUMENTATION:

- Spillet skal have en manual/help page til hvordan alting virker.

SECURITY:

- Username skal være mindst 6 characters
- Username skal være unikt
- Password skal være mindst 8 characters
- Password skal have store og små bogstaver
- Password må ikke indeholde username

MOSCOW krav

I dette afsnit er de ikke-funktionelle krav fra ovenstående afsnit prioriteret ved hjælp af MoSCow metoden.

MUST

- have en GUI
- have en Database
- Have Netværskommunikation
- 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 Enemies
- Have Items
- Have Combat mechanics
- Have End Screen
- Have Character Stats

COULD

- Have Level development
- Have Procedural world
- Have Text Parser
- Have Difficulty
- Have Security

WON'T

- Have Graphics

Accepttest

I dette afsnit opsættes systemets Accepttest. Disse er opdelt efter kategori, først er der accepttest for User stories, og herefter de ikke funktionelle krav.

Funktionelle

Test af User Story 1 - Login 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:

Kommentar:

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:

Kommentar:

Test af User Story 2 - Opret profil 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 datanene for brugerens profil på databasen*
- *Og skærmen skiftes til log ind skærmen*

Resultat:

Kommentar:

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:

Kommentar:

Test af User Story 3 og 4 - Main Menu Scenarie - Tilgå settings

Givet at brugeren er logget ind og har adgang til spillet.

- *Når bruger trykker settings i UI*
- *Så går spillet til settings menuen*

Resultat:

Kommentar:

Scenarie - Start spillet

Givet at brugeren er logget ind og har adgang til spillet.

- *Når bruger trykker "New Game"*
- *Så vises game-interface for brugeren*

Resultat:

Kommentar:

Scenarie - Exit game

Givet at brugeren er logget ind og har adgang til spillet.

- *Når bruger trykker "Exit game"*
- *Så lukker spillet*

Resultat:

Kommentar:

Test af User Story 5-16 - Exit Menu Scenarie - Exit spil

Givet at brugeren har trykket "New Game"

- Når bruger trykker "Escape" på keyboardet
- Så popper et menuvindue op med mulighederne "Resume Game", "Save Game", "Main Menu" og "Settings"

Resultat:

Kommentar:

Scenarie - In game menu - Resume game

Givet at brugeren har trykket "New Game" og derefter har trykket "Escape" på keyboard

- Når bruger trykker "Resume Game" i In game menu
- Så forsvinder menuvinduet og spillet fortsætter

Resultat:

Kommentar:

Scenarie - In game menu - Save – No Combat

Givet at brugeren har trykket "New Game" og derefter har trykket "Escape" på keyboard

- Når bruger trykker på "Save Game" i In game menuen
- Så vises save menuen
- Og en liste af gemte spil

Resultat:

Kommentar:

Scenarie - In game menu - Save – Combat

Givet at brugeren er i combat

- Når bruger trykker "Escape" på keyboardet
- Så kan man ikke se en "Save Game" knap i game menuen

Resultat:

Kommentar:

Scenarie - In game menu - Main Menu

Givet at brugeren har trykket "New Game" og derefter har trykket "Escape" på keyboard

- Når bruger trykker "Main Menu" i In game menu
- Så vises hovedmenuen

Resultat:

Kommentar:

Scenarie - In game menu - Settings

Givet at brugeren har trykket "New Game" og derefter har trykket "Escape" på keyboard

- Når bruger trykker "Settings"
- Så åbnes et vindue med mulighed for konfiguration af spil for bruger

Resultat:

Kommentar:

Scenarie - Exit menu - Settings - Apply

Givet at brugeren har trykket "Settings" og ændret på resolution indstilling

- Når bruger trykker "Apply"
- Så ændres indstillinger som brugeren har ændret

Resultat:

Kommentar:

Scenarie - Exit menu - Settings - Back

Givet at brugeren har trykket "Settings"

- Når bruger trykker "Back"
- Så vises In game menuen

Resultat:

Kommentar:

Scenarie - Exit menu - Settings - Default

Givet at brugeren har trykket "Settings" og har ændret mindst 1 indstilling

- Når bruger trykker "Default"
- Så ændres alle indstillinger tilbage til default settings

Resultat:

Kommentar:

Test af User Story 17 og 18 - Save Menu 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:

Kommentar:

Scenarie - Save Menu - Back

Givet at brugeren har trykket "Save game" fra Settings menu

- Når bruger trykker "Back" i save game menuen
- Så vender spillet tilbage til In game menuen

Resultat:

Kommentar:

Test af User Story 19-22 - Main Menu Scenarie - Main menu - Load Game

Givet at brugeren er logget ind og har adgang til spillet.

- Når bruger trykker "Load game"
- Så vises en liste af gemte spil på profilen

Resultat:

Kommentar:

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:

Kommentar:

Scenarie - Main menu - Load Game - Back

Givet at brugeren er logget ind og har adgang til spillet og trykket "Load Game"

- Når bruger trykker "Back" i load game-menuen
- Så vender spillet tilbage til hovedmenuen

Resultat:

Kommentar:

Test af User Story 23-28 - Spil spillet Scenarie - Spil spillet - Start spillet

Givet at brugeren har trykket "New Game"

- Når bruger anvender piltasterne på keyboard eller trykker på knappen på interfacet til at bevæge sig sydpå fra startrummet
- Så bevæger brugeren sig ind i rummet "Syd" for det rum de står i

Resultat:

Kommentar:

Scenarie - Spil spillet - Enter nyt room

Givet at brugeren har trykket "New Game"

- Når bruger går ind i et nyt rum ved brug af keyboard
- Så giver UI en beskrivelse af rummet spilleren befinder sig i

Resultat:

Kommentar:

Scenarie - Spil spillet - Combat

Givet at brugeren møder en fjende i et rum

- Når bruger trykker "Fight!"
- Så ruller brugeren et tal mod fjenden om at skade fjenden
- Og derefter ruller fjenden et tal om at skade brugeren

Resultat:

Kommentar:

Scenarie - Spil spillet - Combat - Flygt

Givet at brugeren møder en fjende i et rum

- Når bruger trykker "Flee"
- Så flyttes brugeren tilbage til det rum han kom fra
- Og får ikke sit mistede liv tilbage igen

Resultat:

Kommentar:

Scenarie - Spil spillet - Inventory

Givet at brugeren har trykket "New Game"

- Når bruger trykker "Inventory"
- Så vises genstande som bruger besidder

Resultat:

Kommentar:

Scenarie - Spil spillet - Interact

Givet at brugeren har trykket "Start spil" og at der ike er fjender i rummet

- Når bruger trykker "Interact"
- Så kan bruger tage potentielle genstande i rummet til brugerens "Inventory"

Resultat:

Kommentar:

Scenarie - Spil spillet - Level klaret

Givet at brugeren har fuldført det næstsidste rum

- Når bruger bevæger sig ind i sidste rum
- Så får bruger en besked om at spillet er klaret og får mulighed for at gå til "Main Menu"

Resultat:

Kommentar:

Ikke funktionelle GUI

Arkitektur

Table 1

Beskrivelse	Verificering	Resultat	Kommentar
GUI'en skal tilbyde valget mellem 3 resolutions	Visuel		
GUI'en skal kunne være fullscreen	Visuel		
GUI'en skal kunne være windowed	Visuel		

Frontend Arkitektur

Frontend applikationen vil have til formål at håndtere brugerinput og output. Dvs. at der i Frontenden vises det data fra gamelogic, som brugeren skal have, og at det præsenteres på en overskuelig og brugervenlig måde. Dette resulterer i at brugeren kan forstå og kan finde ud af at bruge spillet på den tiltænkte måde. Derudover skal Frontenden tage hånd om bruger input, og sørge for at brugeren giver korrekt input og at der tages hånd om eventuelt forkert input.

Da der er mange forskellige menuer og skærme i spil i frontenden er der lavet følgende C3 model for Frontenden (Figure 6), 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 frontenden selv f.eks. skal der ved Login/Register kontaktes databasen for at få verificeret loginoplysninger, 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. Der skal hertil nævnes at Login og Register står til at snakke med backenden direkte og ikke igennem gamelogic blokken, dette er valgt da funktionen ikke kaldes i gamelogic blokken, men den kaldes direkte i backend controlleren. Derudover er modellen mere overskuelig på denne måde og fungerer bedre til at give overblik over navigationen igennem menuerne.

Pseudo Frontend Arkitektur

(Hovedrapport stuff) 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

(/Hovedrapport stuff) Der er ikke lavet sekvensdiagrammer for alle af projektets userstories, da mange af disse fungerer på samme måde og derfor ikke bidrager med noget nyt ift. dokumentationen. Nedenfor ses pseudo sekvensdiagrammer for de fire userstories:

- Login
- Register
- Save Game
- Load Game

Først ses "Login" (Figure 7), som viser forløbet af userstory "Login", med en reference

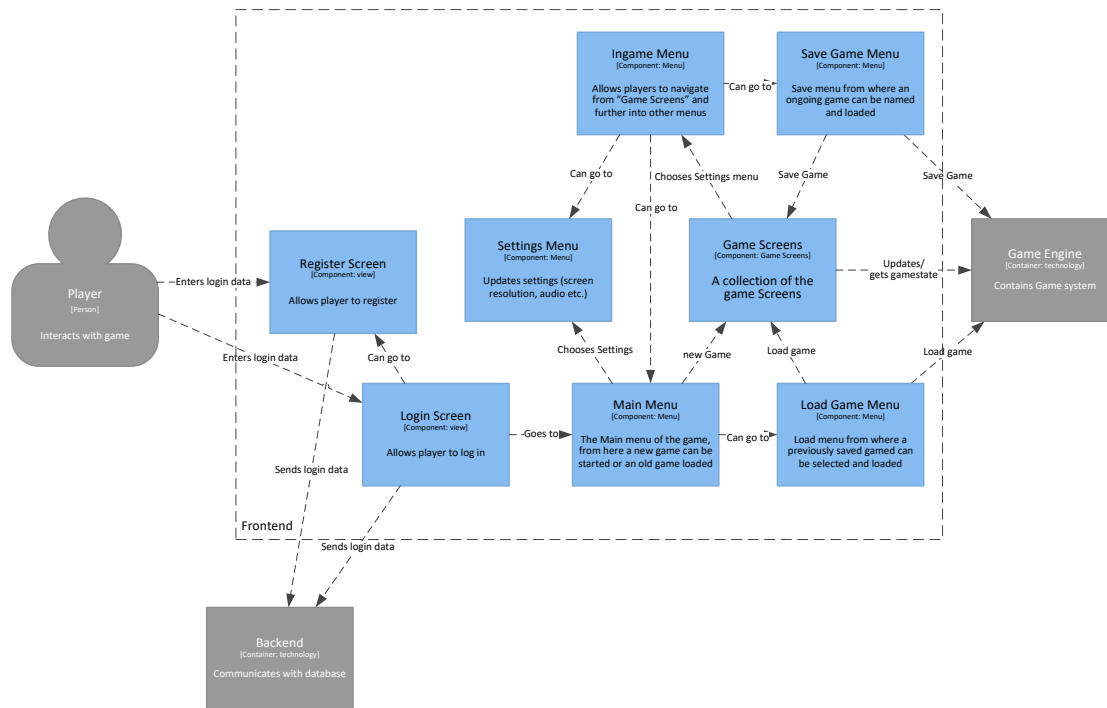


Figure 6: C3-Model for Frontend. Modellen fortæller hvordan man kan navigere igennem forskellige menuer og hvilke menuer der kan føre til hvad. Derudover kan man se hvilke blokke der snakker ud af frontend og sammen med resten af systemet.

til userstory "Register", hvis brugeren ikke er registreret i forvejen. Udover dette er der ydermere vist håndtering fejlet login. Det skal her nævnes at brugeren kan ikke spille spillet uden at logge ind først, der tillades ikke at spillet kan spilles i Offline-tilstand.

Dernæst ses "Register" (Figure 8), som viser forløbet af hvordan en bruger kan registrere sig med en profil i systemet. Der kan ses på Figure 7, at hvis en bruger ikke er oprettet skal man gøre dette først. Derefter skal man vælge et brugernavn og kodeord, hvis brugernavnet er ledigt og alt ellers går godt, kommer man tilbage til "Login" skærmen. ellers får man en fejlbesked på skærmen og bliver bedt om at prøve med et andet brugernavn.

På Figure 9 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 selv kald hver gang skærmen skiftes. 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.

Til sidst kan der på Figure 10 ses "Load Game", som viser forløbet når en bruger gerne vil hente et tidligere gemt spil fra databasen, set fra Frontends perspektiv. Denne user-story minder på mange måder om "Save Game", men i stedet for at sende et element til

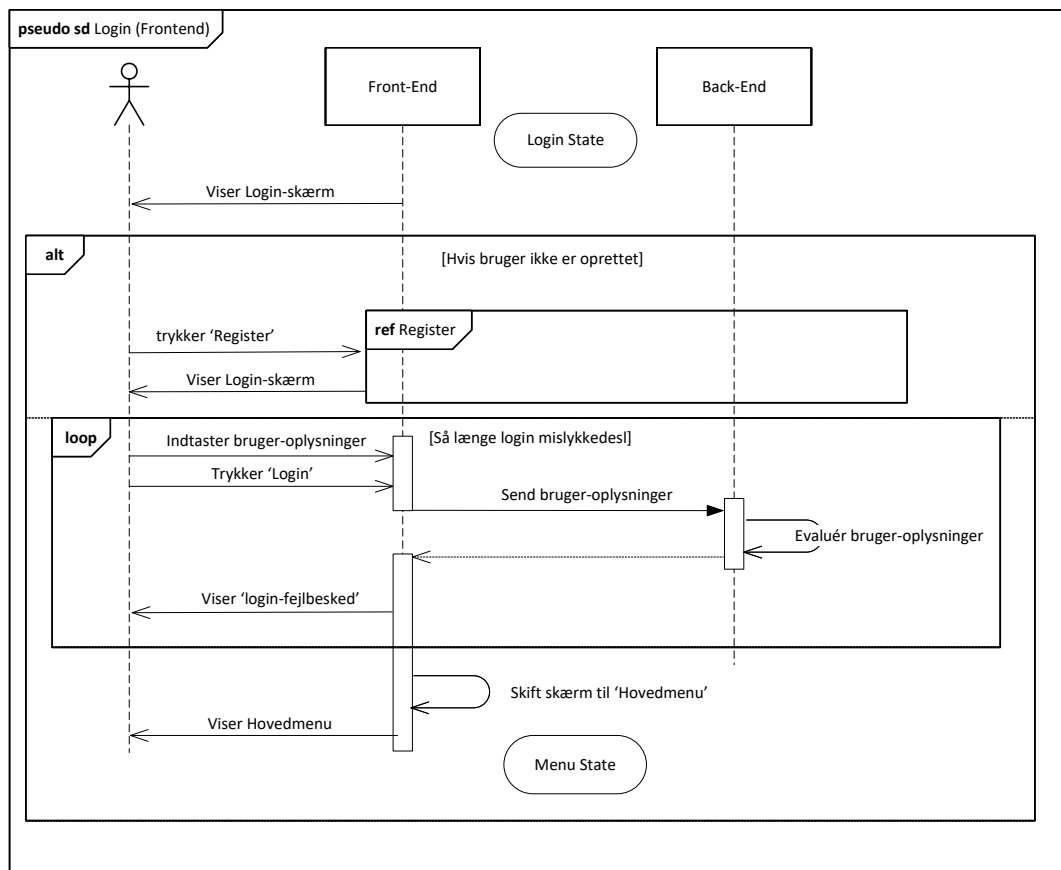


Figure 7: Pseudo sekvensdiagram af forløbet af userstory "Login", set fra Frontends perspektiv. Med reference til "Register" userstory og håndtering af forkerte login oplysninger.

databasen, skal der hentes et gemt spil fra databasen, med alle de data der skal bruges for at kunne load sit spil op præcis som man forlod det.

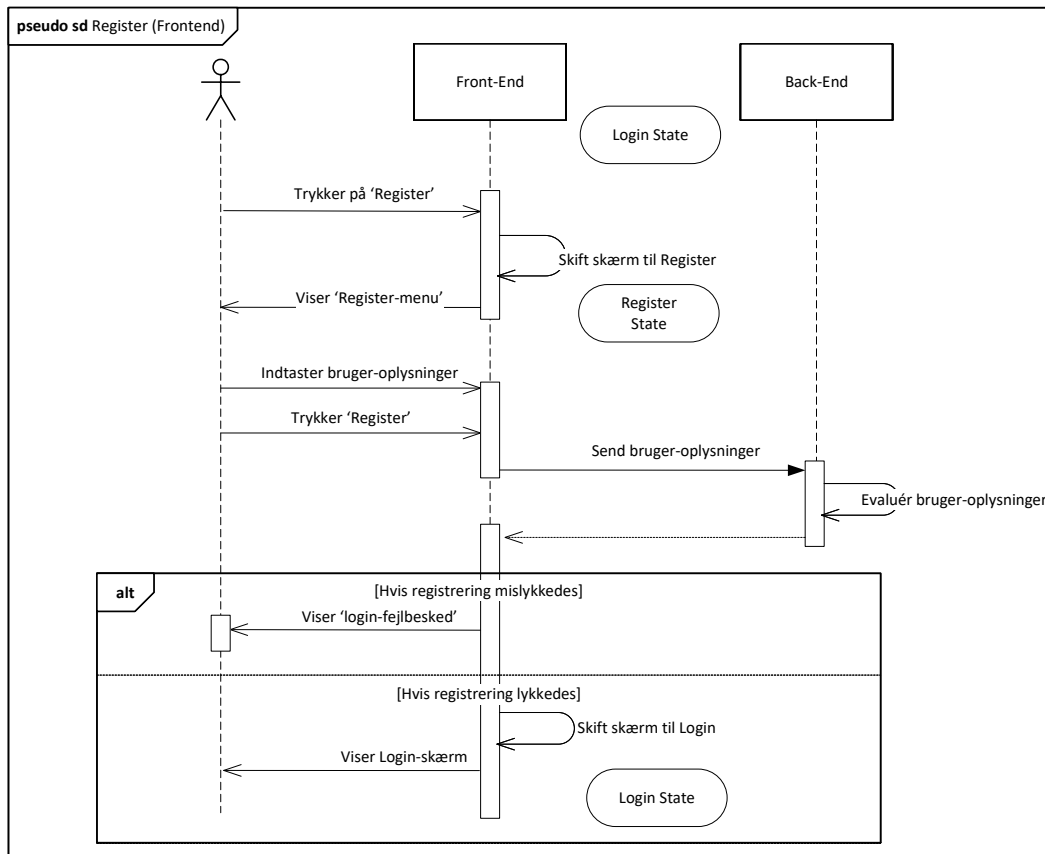


Figure 8: Pseudo sekvensdiagram af forløbet af userstory "Register", set fra Frontends perspektiv. Med håndtering af forkerte login oplysninger.

Database Arkitektur

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.

Måden kommunikationen vil foregå igennem systemet vises her. Her ses der at en bruger interagerer med Front-End, data går videre til Game Module, som kommunikerer med Back-End, hvor der til sidst enten bliver skrevet til databasen eller hentet data fra databasen. For at illustrere databasens funktionalitet er der blevet dannet sekvensdiagrammer, som viser hvordan databasen vil kunne gemme et spil og hente et gemt spil.

For at illustrere databasens funktionalitet er der blevet dannet sekvensdiagrammer, som viser hvordan databasen vil kunne gemme et spil og hente et gemt spil.

I 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

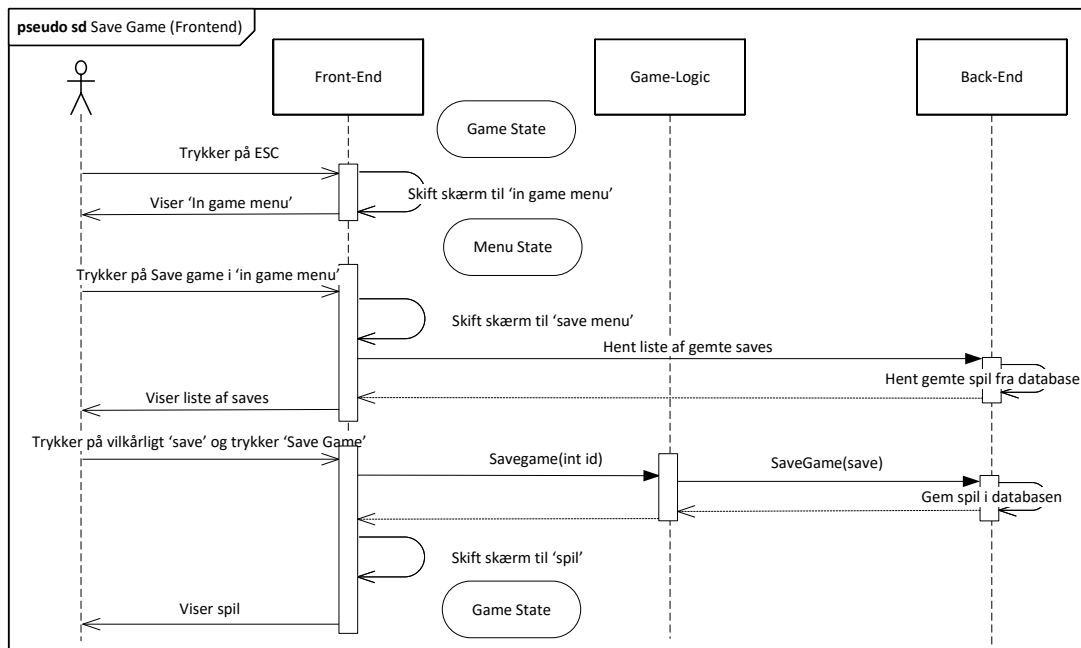


Figure 9: Pseudo sekvensdiagram af forløbet af userstory "Save Game", set fra Frontends perspektiv. Der laves 2 kald til databasen igennem Backend, hvori der i det første kald, "Hent liste af gemte saves" hentes en liste af brugerens gemte spil og i andet kald gemmes brugerens nuværende spil henover det valgte spil.

databasen til DAL og så fra DAL til SaveController.

I diagrammet GetAllSaves foregår kommunikation på samme måde som ved at hente rum beskrivelser. Forskellen her er dog at der i stedet for hentes en list af alle gemte spil.

I diagrammet GetSave ses der hvordan et specifikt gemt spil hentes. Her ses der at det igen går først igennem DAL. Et gemt spil vil have et ID, som der kan anvendes til at finde de korresponderende værdier som dette gemte spil har. Her ses der at der vil findes items, enemies, puzzles og hvilke rum en spiller har besøgt. Dataene returneres herefter til DAL og så til SaveController.

I dette diagram, SaveGame, ses der hvordan et spil vil gemmes. Måden dette vil fungere på er at spilleren vil starte med fem tomme gemte spil, som bliver seeded til databasen. Dette bliver gjort for at begrænse en bruger til fem gemte spil. Så for at gemme et spil findes der først hvilket gemte spil der skal overskrives. Derefter findes de relevante værdier i dette gemte spil. Dernæst slettes disse værdier og der tilføjes de nye korrekte værdier.

DAL Arkitektur

I dette segment vil der blive forklaret tankerne bag arkitekturen vedr. oprettelsen af en funktionel database, som kan anvendes til at opbevaring af data til dette projekt. Til oprettelsen af en funktionsdygtig database, i dette projekt, kræves der et relativt tæt sammenhold mellem databasen og backend api'en. API'en er ansvarlig for kommu-

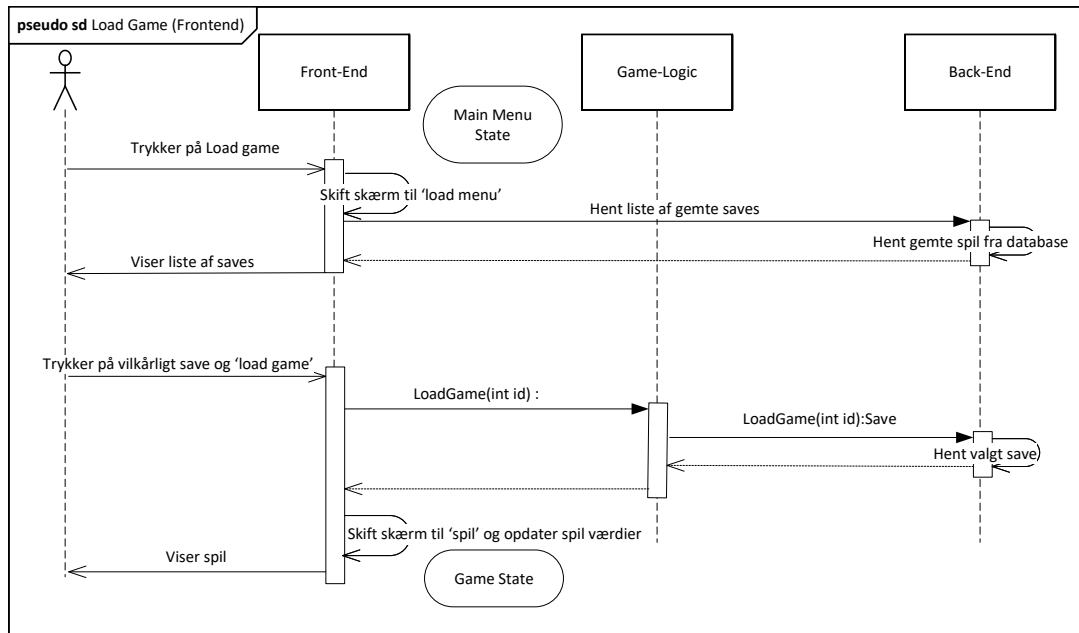


Figure 10: Pseudo sekvensdiagram af forløbet af userstory "Load Game", set fra Frontends perspektiv. Der laves 2 kald til databasen igennem Backend, hvori der i det første kald, "Hent liste af gemte saves" hentes en liste af brugerens gemte spil og i andet kald hentes brugerens valgte spil og spillet startes op.

nikation mellem Front-end, Game Module og databasen.

Når data enten skal sendes til eller hentes fra databasen så er det igennem et DAL. Systemets DAL giver os en simplificeret adgang til dataene gemt i databasen, og fungerer som en mellemmand for systemet, da alt kommunikation til databasen går igennem den. I denne DAL har vi en Authentication, som bliver anvendt når en bruger logger ind, og når der oprettes en ny bruger. Denne blok kontrollerer brugernavn og kodeord, som sendes og hentes i databasen. Ydermere vil der ikke sættes fokus på sikkerhed, som ses i kravene stillet for systemet. Yderligere vil det også være muligt at både gemme et spil og indlæse et spil. Begge af disse vil operere på det samme data, dog ville den ene, LoadGame, hente data fra systemets database, og den anden, SaveGame, vil sende data til systemets database. LoadGame i DAL er ansvarlig for at hente spillerens data fra databasen, såsom hvilket rum de var i og mængden af liv de har tilbage. Hernæst er der SaveGame. Denne indeholder funktionaliteten for at sende et gemt spil, altså dataene for spillerens nuværende spil. Heri vil der også gemmes information og spillerens nuværende tilstand i spillet. Det ville f.eks. være rummet som spilleren er i når spillet bliver gemt. Begge af disse vil være ansvarlige for at håndtere data som er individuelle for hver spiller og hvert gemt spil. Udenfor systemets DAL ville der også være inkluderet Models i konstruktionen af systemets database. Models vil indeholde de entities som Authentication og Load-/SaveGame vil bestå af. Yderligere vil Models også være ansvarlig for forholdene imellem de forskellige entities.

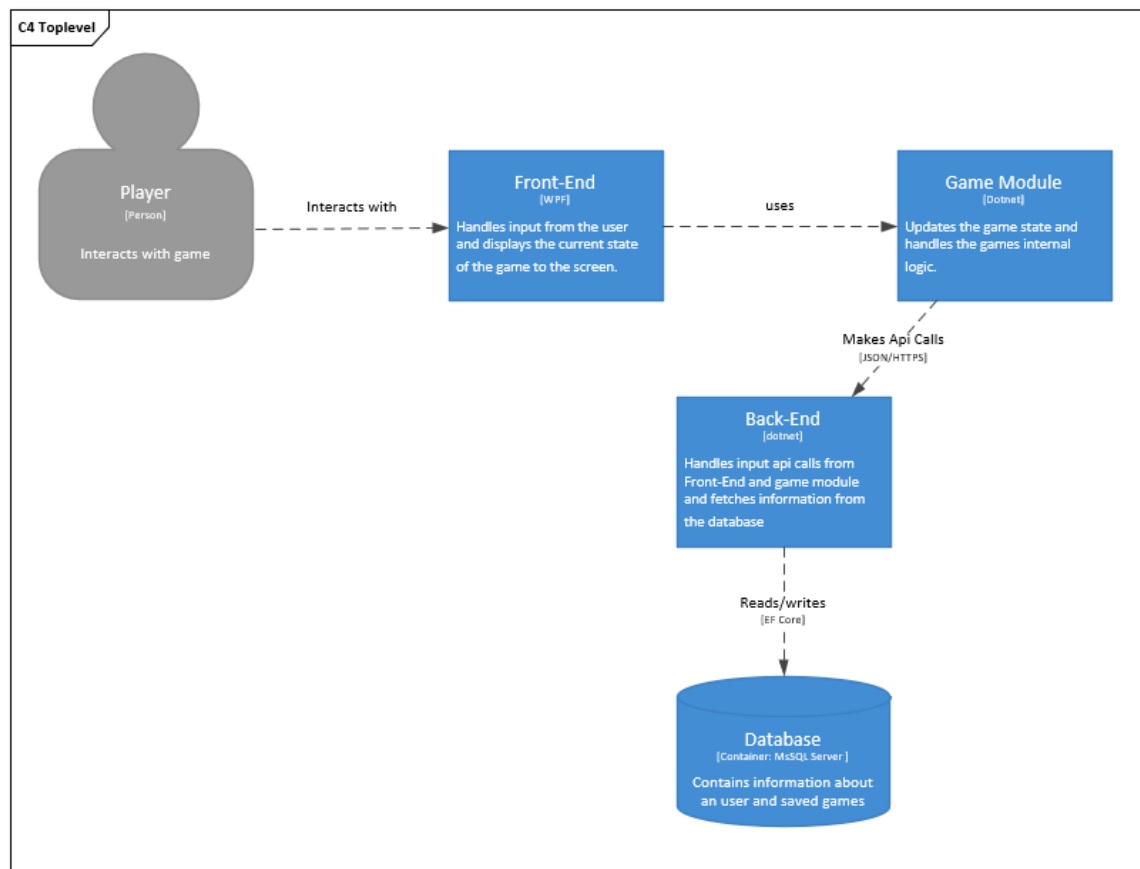


Figure 11: C4 top level diagram, som viser kommunikation mellem systemets segmenter

Design

Overordnet System Design

Frontend Design

I frontenden ønskes det at hele spillet foregår i et vindue. Det er derfor nødvendigt at programmet kan skifte mellem forskellige views uden at skulle åbne et nyt vindue, hver gang der skiftes. Løsningen til denne udfordring beskrives nærmere i implementerings afsnittet, nemlig subsection 4.4. Spillet vil bestå af en række af vinduer, som giver spilleren den nødvendige information for at de kan spille spillet (så som at logge ind, gemme og hente spil, samt spille spillet).

Inden arbejdet på Frontend arkitekturen begyndte, er der blevet lavet en teknologiundersøgelse (INDSÆT REFERENCE HER), om hvilket udviklingsværktøj Frontenden 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 (INDSÆT REFERENCE HER), hvor flere fordele og ulemper for både unity og .NET frameworket er sat op.

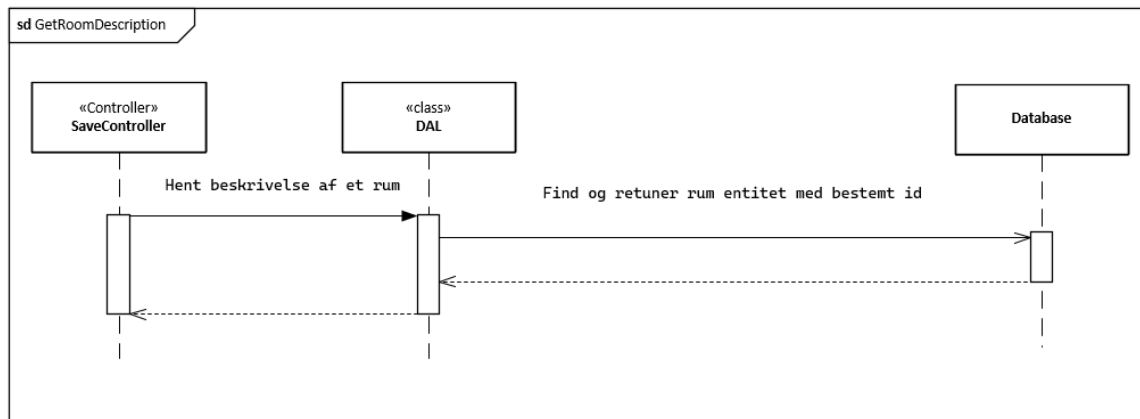


Figure 12: Sekvensdiagram for hvordan rum beskrivelser vil blive hente fra databasen af SaveController

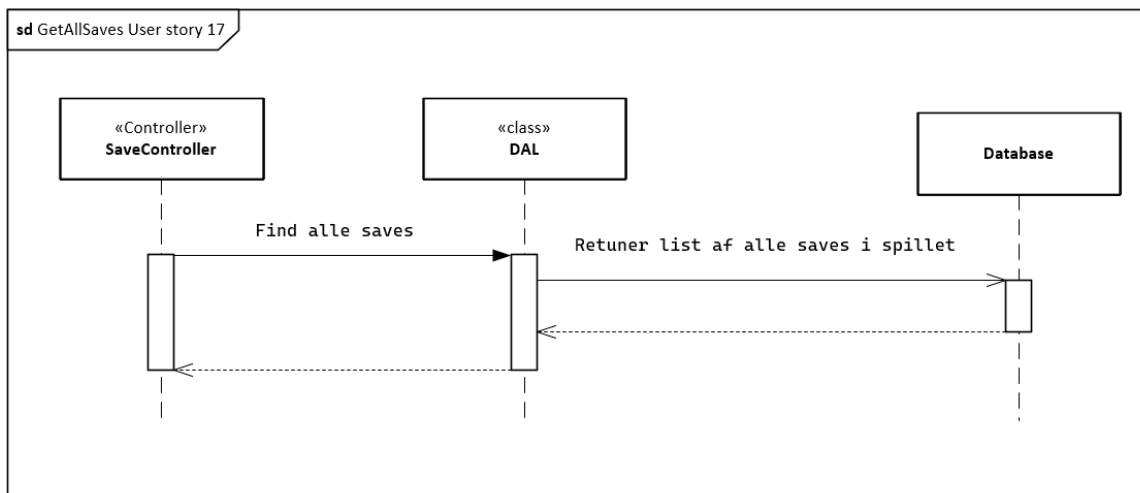


Figure 13: Sekvensdiagram for User story 17: GetAllSaves, i relation til hentet data fra database

Her følger en række af mockups¹ af nogle af spillets views.

Room

Room view (Figure 18) er det primære spil-vindue. 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

¹Lavet i Unity

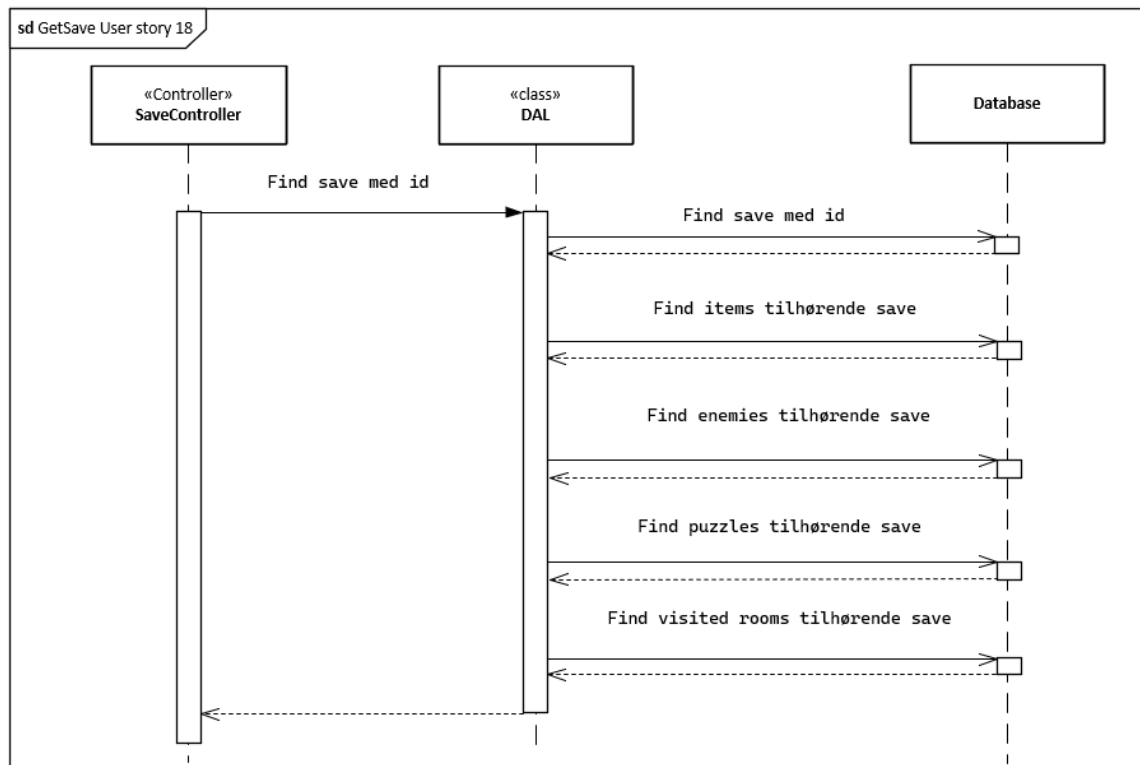


Figure 14: Sekvensdiagram for User story 18: GetAllSaves, i relation til hentet data fra specifikt gemt spil fra database

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.

Combat

Combat vinduet er baseret på room vinduet. Strukturen er den samme: der er et kort øverst til højre, en beskrivelse øverst til venstre og knapper neders til højre. Nederst til venstre er der information om hvordan kampen går, i stedet for en liste af elementer i rummet.

Knapperne består af nogle "menu" knapper, som lader dig gå til spil menuer, samt en 'Fight' knap og en 'Flee' knap. 'Fight' knappen lader spilleren kæmpe mod fjenden, mens 'Flee' knappen lader spilleren flygte fra kampen og tilbage til rummet som spilleren kom fra.

Login

Når spillet startes bedes spilleren prompte at logge ind på deres profil. Dette sker i login vinduet. Spilleren kan indtaste sit brugernavn og kodeord i de to tekstfelter 'Username' og 'Password'. Knappen login fører dem videre til spillet, hvis det indtastede login er korrekt.

Knappen create user fører spilleren til et vindue som ligner login vinduet, og som lader

spilleren oprette en ny bruger.
Exit lukker spillet.

Settings

Settings viduet tillader at spilleren kan ændre indstillinger for spillet, og kan tilgås fra hovedmenuen, samt fra selve spillet.

Det er her muligt at ændre f.eks. skærmopløsning og lydstyrke. Det er muligt at gemme de indstillinger som er valgt, forlade menuen uden at gemme de valgte indstillinger, samt gendanne standard indstillingerne for spillet.

Database Design

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 grupper 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.

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 undtagelse af en combat, og gemme spillet. Det skulle derefter være muligt for spilleren at load spillet igen, hvorefter spillet er i samme stadie som man gemte det i.

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 spillets backend. En spiller skal derefter kunne gemme unikke 5 spil med forskellige oplysninger. Håndteringen af restriktionen med max 5 forskellige spil pr. Bruger, håndteres ved at oprette 5 standard gemte spil pr. bruger som vi overskriver, når vi gemmer. Der kan på denne måde ikke oprettes mere en 5 gemte spil pr. bruger. I et gemt spil ønskede vi at gemme en række forskellige attributter for spilleren. Første og fremmest får hver spil et unikt id som vi benytter til 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 forskelligt fra de 4 andre gemte spil som tilhører brugeren. En spiller 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.

Tabellen med inventory har 2 atributter, et ID, som svarer til en bestemt genstand, og en reference til set 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 besøgt rum id. Denne parring er unik da man blot behøver at besøge et rum før det er synligt på kortet.

Der er i projektet oprettet klasser tilsvarende ER diagrammerne.

Frontend Implementering

Det endelige design af vinderne følger tæt det oprindelige mockup design. Der benyttes MVVM (Model, View and View Model) designpatterns Det har undervejs i implementeringen vist sig et behov for væsentlig flere menuer end først antaget, og disse er implementeret efter samme overordnede design, som resten af systemet, så derved følges stilen og følelsen af spillet.

Det endelige design har følgende views: Login, Register, Main, Settings, Ingame, Load, Save, Inventory, Character, Victory, Defeat, Room og Combat.

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 controlls, 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 skiftet. Dette resulterer i et mere flydende User Interface for brugeren og

derived en alt i alt bedre oplevelse.²

Der er implementeret at nogle af spillets funktioner kan tilgås via key-bindings, hvilket har nødsaget et brud på MVVM-designet. Når mediatoren skifter mellem views bilver fokus ikke sat til indholdet af det nye view, og keybindings virker derfor ikke. For at løse dette sættes top-elementet på den nye side som fokus. Dette kan dog ikke gøres i MVVM, så her er det pattern brudt, da det er nødvendigt at gå ind i code-behind filen for at sætte fokus.

De følgende afsnit viser et udvalg af view, samt en beskrivelse af hvordan de afviger fra det oprindelige design og en beskrivelse af interessante programmerings-tekniske beslutninger.

Login view

De overordnede struktur af login (Figure 24) og register menuerne følger meget tæt det oprindelige design. Alle elementer er blevet stiliseret så de matcher det ønskede look. Dette er gjort ved brug af globale resources og styles i app.xaml filen (INDSÆT REFERENCE HER). Dette gør det nemt at oprette nye views og ændre udseende af hele spillet. Selve login håndteres af backenden som kaldes af login og register knapperne via command bindings.

Room View

Visuelt er room view (Figure 25) 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 separat view), som kan tilgås ved at trykke på Inventory knappen.

Alt tekst er vist med data binding.

Combat View

Combat view (Figure 26) er bygget med room view som skabelon, så de fleste elementer er ens. Knappernes antal og funktion er ændret, og sammenlignet med det oprindelige design er knapperne for items og character fjernet, da det blev besluttet at det ikke skulle være muligt at tilgå sit inventory under en kamp. I stedet for en beskrivelse af rummet og en liste af items vises der nu en beskrivelse af hvordan kampen går nederst i venstre side af skærmen. Tal-værdierne hentes fra gameengine, og sættes ind i en tekststreng, som gør det nemt for brugeren at forstå hvordan det skal fortolkes. Der er yderligere tilføjet en healthbar, som giver en visuel indikation af, hvor tæt spilleren er på at tabe spillet og skifter farve fra grøn til gul til rød, som spilleren tager mere skade. Baren giver derfor lidt farve til et ellers meget gråtonet spil.

Settings view

I settingsmenuen (Figure 27) er det muligt at vælge lydstyrke for musikken i spillet (samt slukke helt for musikken) og vælge mellem tre skærmopløsninger (1280x720, 1920x1080 og 2k). Det tre valgmuligheder er valgt til at teksten på skærmen stadig er

²<https://www.technical-recipes.com/2018/navigating-between-views-in-wpf-mvvm/>.

læselige. Kortet i Room og Combat view skalerer med skærmopløsningen, men sætter en nedre grænse på 300x300 pixel. Tekststørrelsen gør dog at den praktiske nedre grænse, hvor spillet ser ud som det skal, er 1280x720. Ved skærm opløsning større end 2k bliver teksten og kortet for småt til at kunne ses ordentligt, hvorfor 2k er en naturlig øvre grænse for skærmopløsningen. Alle opløsninger imellem de to grænser burde være i orden (så længe det bare nogenlunde følger en 4:3 eller 16:9 ratio).

For at sørge for at den valgte skærmopløsning er brugt i alle views er der oprettet et objekt til at holde informationer om blandt andet indstillinger, samt andre informationer, som det ønskes at kunne tilgå fra flere forskellige views uden at være nødsaget til at sende data med rundt, når der skiftes mellem views. Dette objekt følger et singleton design pattern, og den samme instance af objektet kan derfor tilgås fra alle de view som skal bruge informationer derfra, på den måde opnås det at der kan sættes glodbale indstillinger som kan bruges på alle views uden at informationen skal sendes med i et skærmskift.

Settings menuen kan tilgås fra både main menu og ingame menu, og det er derfor nødvendigt at spillet ved hvilken menu brugeren kommer fra, sådan at brugeren kan komme tilbage til den rigtige menu, når settings menuen forlades, ved at der trykkes på back-knappen. Til dette bruges singleton objektet fra tidligere afsnit også, da det gør det nemt at bringe informationen mellem views. Dette bruges også i ingame menu, da denne kan tilgås fra både room view og combat view og skal kunne returnere brugeren til det rigtige view, når brugeren vælger at starte spillet igen.

Load view

Load (Figure 28) og Save menuerne præsenterer spilleren med en liste af gemte spil, som brugeren kan hente, eller gemme. Dette opnås ved at der hver gang spilleren åbner en af de to menuer, hentes en liste af tilgængelige 'save-games', som via databinding, præsenteres for brugeren. Når der trykkes på Save/Load sendes den fornødne kommando til backenden og backenden tager fat i databasen og udfører enten save eller load kommandoen.

Note om Baggrundsfarver

Da den generelle baggrundsfarve i spillet er sort er alle spillets interface elementer oprettet så baggrundsfarven matcher. Dette er for det meste nemt opnået ved brug af WPF styles, men enkelte elementer (så som resolution dropdown menu, brugt i settings menuen) viste sig at være betydeligt mere omfattende. Det viser sig at det valgte element (WPF combobox) ikke tillader at baggrundsfarven for dropdown elementerne ændres i Windows 8 eller senere. Løsningen er at lave en kopi af hele template (ca. 300 linjer kode) for combobox og ændre 3-4 linjer.³

³<https://social.technet.microsoft.com/wiki/contents/articles/24240.changing-the-background-color-of-a-combobox-in-wp>

Implementering

System Implementering

Under implementeringen af nogle funktioner i samtlige moduler, er funktioner blevet lavet til funktioner der følger følgende opstilling:

```
public async Task<T> FooBar();
```

Funktionerne returnerer en Task af typen T og den 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.

Game Engine

Game Enginen er skrevet i C#, et objektorienteret programmeringssprog med stærkt library support der tillader udvikleren at fokusere på udvikling af applikationer i stedet for udvikling af libraries til at støtte projektet.

Nedstående præsenteres en diagram over de mest kritiske komponenter Game Enginens interne spil logik og deres relationer til hinanden. Der er ikke medtaget interfaces, eller klasser som er map, items, BackendController og logs som er mere specifikke til spillet og ikke til den interne logik.

Gamecontroller Implementering

GameControlleren er den centrale komponent i game enginen, denne er ansvarlig for kommunikation til frontend del af applikationen. I denne implementering af GameControlleren har den adgang til alle spillets funktionaliteter gennem dennes association til Combatcontroller, se Figure 29, og BackendController, som ikke er vist på Figure 29.

Fra et implementering perspektiv er dette en nem løsning for en lille applikationen som denne men fra et design perspektiv er dette en dårlig løsning. GameControlleren har alt for mange grunde til at ændre sig og følger næppe SOLID principperne.

GameControlleren saver og loader spillet gennem sin relation til BackendController. Desværre er load funktionen ret kompliceret og burde være delt op i mindre funktioner. Source code for load kan ses på Figure 30

Koden bliver kompliceret idet den forsøger at gennemløbe alle Room objects i spillet for på korrektvis at fjerne enemies og items, som allerede er blevet samlet op tidligere. Her gennemløbes alle Room objects og i hvert room gennemløbes alle items i Room chest objectet for at krydsreferere alle items med inventory listen for at se om de skal fjernes som en del af save gamet.

Ligeledes håndterer den enemies og krydsreferer alle enemies med slainEnemies listen for at se hvilke enemies spilleren allerede har besejret. Disse enemies fjernes herefter fra spillet.

Generering af Map

Map klassen generer et map til spillet. Denne består af en simple liste af lister over, hvilket rooms hvert room er forbundet til.

Denne Process er kompliceret og kræver en uddybende forklaring. Problemet består i at afgøre hvordan man sikre at det samme map bliver genereret hver gang spillet loades.

Gem en map layout fil

Denne løsning viste sig at være den bedste løsning for at sikre sig, at map layoutet kun skulle eksistere i en enkel folder i projektet. Men det viste sig vanskelig at læse en sådan fil uafhængigt af pc'en programmet blev kørt på.

Lav en klasse som generer map layout filen

Ultimativt blev dette løsningen, som benyttes i projektet. En MapCreator klasse danner en map layout fil når dens konstruktor bliver kald.

En map klasse som store map layoutet kan nu læse layout filen og danne et map ud fra denne fil. Filen består af linjer med formen "*leftRoomId, TopRoomId, RightRoomId, BottomRoomId*". Den først linje dækker room 1, næste linje dækker room 2 osv.

Hver linje i map layout filen mappes til en liste af roomId'er, der kan insertes i Map klassens mapLayout listea. Denne Liste danner grundlaget for spillets map og diktere hvordan spilleren kan navigere i spillet. Denne mapping mellem strings og int array er vist i Figure 31.

Items og Enemies

Som det sker for rooms, ligeledes sker der for enemies og items. MapCreator klassen generer separate filer for enemy positions og item lokationer. Map klassen kan herefter læse filen og mappe hver linje i filen til, en enemy eller item og placere det i det korrekte room.

Log

Kommunikation med front end fra GameControllern og CombatControlleren gør brug af en log. Logen består af et dictionary datastruktur som GameControllern og CombatControlleren benytter til at logge vigtige infomation, som frontend kunne have brug for at vise til spilleren.

Dette kan være infomation om spillerens bevægelse fra et Room til et andet. Det kan også være infomation omkring spillerens eller enemies tilbageværende livsmængde osv. Dette isolere frontend fuldt fra Game Engine, da frontend ikke har andre accesspunkter til infomation om hvad der sker i spillet.

Derved opnår frontend separation of responsibility da frontend nu kun er ansvarlig for at display infomation som det er givet af game enginen.

Kommentar til implementeringen

Gennem udviklingen af Game Enginen er der brugt interfaces eller abstrakte klasser til implementering af alle funktionaliteter. Dette gør det både nemt at teste implementeringerne sam at udvide spillet. Men for scopet af opgaven er dette nok "Overkill". Det er godt at bruge men for en applikation så lille som denne, der ikke kommer til at udvide sig er det nok for meget boilerplate kode.

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 Gnoblings" 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 inkludere automatiseret unittests, integrationstest og accepttest.

Modultest Frontend

Modultesten af Frontenden er lavet i meget tæt samarbejde med Game Engine holdet, 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.

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 på. Var det derimod en ændring der inkluderede databinding til Game Enginen, så 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.

Eksempel på frontend test i forbindelse med Game Engine

Et godt eksempel på hvordan frontend testene blev kørt er ved implementeringen af Room-viewets map element og mere specifikt bevægelsen fra et rum til et andet, altså de implementerede "Go North,East,South,West" knapper, som krævede både en visuel og funktionel del.

Den visuelle del bestod i at kortet skulle opdateres, spilleren flyttes og rum-beskrivelsen opdateres.

Den funktionelle del bestod i at Game Enginen skulle kontaktes på korrekt vis med rigtige parametre, de rigtige databindings skulle opdateres, således at den visuelle del blev opdateret korrekt og derefter skulle informationen om det rum gemmes korrekt i Game Enginen.

For at være sikker på at alt data blev opdateret korrekt blev programmet kørt i debug mode, hvor der kunne sættes break-points i koden og værdierne på diverse variabler, såsom roomdescription og currentroom kunne undersøges. Samtidig blev der kigget

på den visuelle del af selve viewet, hvor der blev tjekket om beskrivelsen blev korrekt displayet, mappen opdateret korrekt og at spilleren flyttes korrekt.

Eksempel på frontend test

Ikke alle moduler krævede at Game Enginen blev kontaktet. Eksempelvis Mediatoren, som er beskrevet i Frontend implementings afsnittet som kan findes her: subsection 4.4. Med dette modul blev der lavet mocks, hvor vi satte en dummy knap op til at kunne kalde notify() funktionen i Mediatoren og der blev tjekket visuelt om viewet blev skiftet uden at hele applikationsvinduet blev skiftet og at det korrekte view blev vist.

Modultest Game Engine

Game Engine styrer spillets indre logik. Dette dækker over alt fra spillerens bevægelse gennem spillet til "save" og "load" af nye og gamle spil. Det er derfor yderst nødvendigt at denne er fuld testet.

For at sikre Høj kvalitet er Game Engine skrevet med Robert C. Martins "Three Laws of TDD" [CleanCode] i baghoved hvilket fører til at Game Engine er eksklusivt test ved hjælp af black-box testing. Alle testene tester kun det offentlige interface, som er gjort tilgængelig.

GodkendelsesTabel Game Engine

Nedestående præsenteres en fuld tabel for alle classes testet som led i Game Engine. De vigtigste er GameController, CombatController, Player/Room 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 GameControllerrens ansvars punkter.

Table 2: 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	<ol style="list-style-type: none"> 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 	<p>Game Controller Har 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 fordi der ikke er skrevet nogen specifikke test til dem Fejler GameControllern sin komponent test.</p>	FAIL

Combat Controller	<ol style="list-style-type: none"> 1. Kan Håndtere Combat Rounds 2. Kan håndtere at Player løber væk fra Combat 	<p>Combat controller kan håndtere at spilleren løber fra combat og at Player engager i combat. CombatController can 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 "critical hits" hvis dice rolleren slår 20.</p>	OK
DiceRoller	<ol style="list-style-type: none"> 1. Kan emulerer et kast med en N siddet terning 2. Kan emulerer N kast med en M siddet terning 	<p>DiceRoller Kan emulere et eller flere terninge kast med samme antal sidder. Denne kan ydmere stille krav for at fordelingen af disse terningekast har en normal distribution og dermed er alle udfald lige sandsynlige.</p>	OK

BaseMapCreator	1. Kan generere et map layout file	BaseMapCreator kan på korrekt vis generere et map layout file, den kan ydmere generer item layout files og Enemy layout files, der hjælper Map klassen med at genererer spillet Map. Der ikke skrevet test for eksistensen af item og enemy layout Map og derfor kan der ikke stille garanti for at disse bliver genereret på korrektvis. Testen Fejler derfor.	FAIL
BaseMap	1. Kan anskaffe Rooms ud fra en Direction	Map kan anskaffe Rooms ud fra en given Direction. Map kan på korrektvis generer et map ud fra mappets layout file. Den kan også på korrektvis finde udaf hvilke Rooms har adgang til hinanden. Der er ikke skrevet test for at bekræfte at enemy og items er i de korrekte lokationer baseret på enemy og item layout filerne. Derfor fejler denne sin Test.	FAIL

Player	<ol style="list-style-type: none"> 1. Kan angribe Enemy 2. Kan tage skade 3. Kan skade Enemy (Ikke Kritisk) 4. Kan skade Enemy (Kritisk) 	Player kan angribe, skade og tage skade fra enemy.	OK
Enemy	<ol style="list-style-type: none"> 1. Kan angribe Player 2. Kan tage skade 3. Kan skade Player (Ikke Kritisk) 4. Kan skade Player (Kritisk) 	Enemy kan angribe, skade og tage skade fra Player.	OK
Log	<ol style="list-style-type: none"> 1. Kan log et event 2. Kan anskaffe et event 3. Kan merge to logs 	Log kan logge et event, anskaffe eventet og merge to forskellige logs.	OK
Room	<ol style="list-style-type: none"> 1. Kan tilføje Player 2. Kan tilføje Enemy 3. Kan fjerne Player 4. kan fjerne Enemy 	Room kan fjerne/tilføje Player og Enemy som forventet	OK

Items		Items så som sword, shield, axe osv. indeholder kun data og har derfor ikke nogen testbar adfærd andet end deres constructor. De kan alle konstrueres på korrekt vis.	OK
--------------	--	---	-----------

Mock testing

I nogle tilfælde kan det være umuligt at lave troværdige tests, når en klasse f.eks. Player er dyb afhængig af DiceRoller klassen. Dette skyldes at DiceRoller danner pseudo-random outputs og derfor er test med den ikke altid troværdige.

Her benyttes mock tests og dependency injections til at sikre at DiceRoller klassen danner de samme outputs hver gang en test køres. Derved kan alle scenarier for Player klassen testes, hvor man kan regne med at DiceRoller giver et bestemt output.

Test Resultater for Game Engine

Nedestående vises kort resultatet for Game Engine test, når de alle køres via visuel 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.

Discussion af Test Result

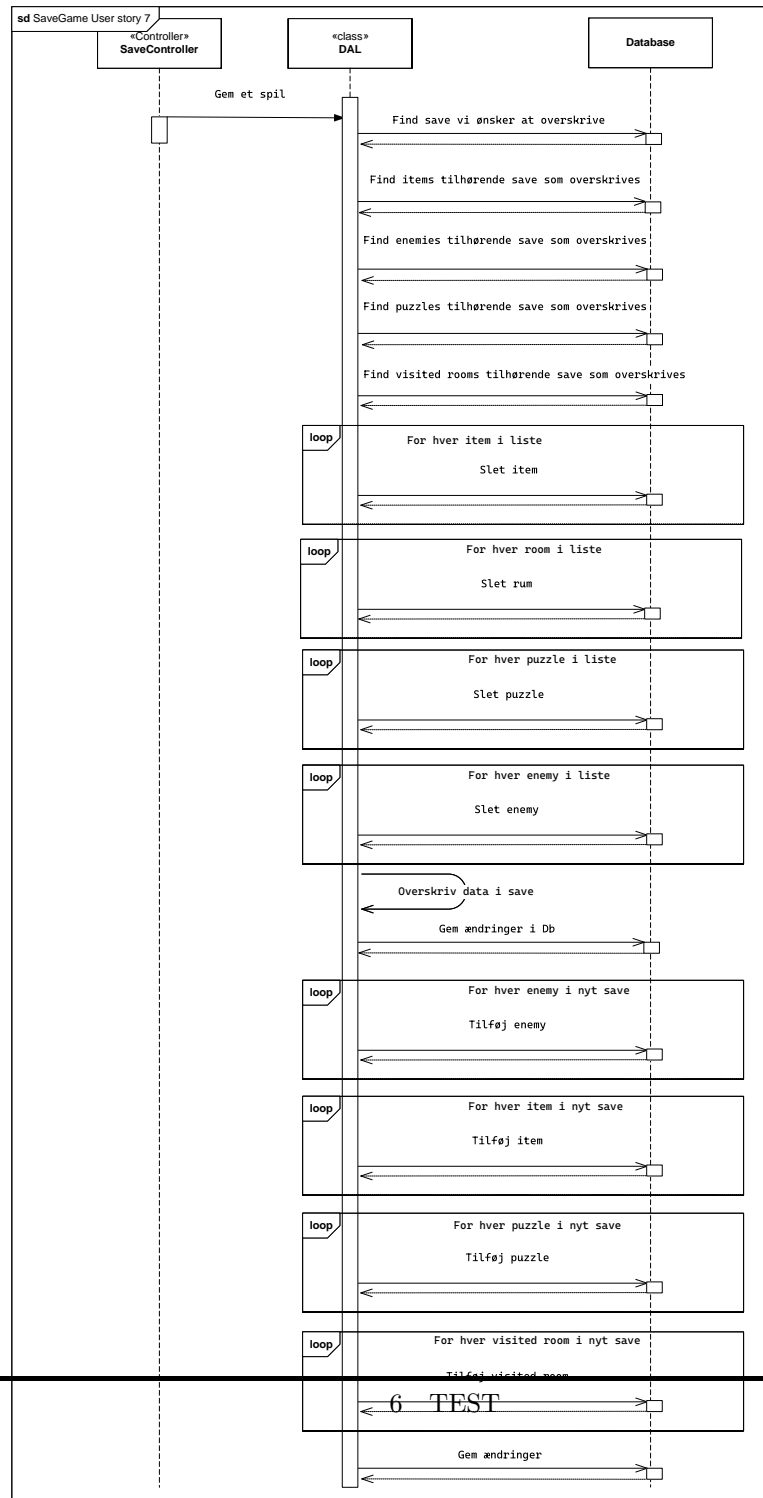
Projektets implementering og kravspecifikationerne har produceret adskillige tests, som projektet i overvejende grad har bestået. De fleste features er blevet testet i henhold til kravspecifikationerne se (REF) og har produceret resultater, der indikerer at hver feature fungerer som ønsket.

Alle funktionelle tests er beskrevet med user-stories og er evalueret med et “Godkendt” eller “Fejl” (REF Here) og evt. en forklarende kommentar. Ikke funktionelle test er har fået en evaluering “OK” eller “FEJL” og kan ses mere detaljeret i (REF HERE). Langt størstedelen af alle tests, for produktet, 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.

To eksempler er “puzzles” og “Delete Save Game”, der skulle have været implementeret, som en del af det færdige spil. Grundet tidspres er disse features ikke blevet implementeret, men i stedet for, er de blevet nedprioriteret til fordel for andre features, såsom “Combat” der er blevet vurderet mere essentiel. “Delete Save Game” er ikke blevet implementeret som specificeret i kravspecifikationerne men eksisterer i stedet for, som evnen til at overskrive allerede eksisterende save games. Her er kravende ikke blevet opdateret til at reflektere den anderledes implementering

Den stores success rate skyldes en insistens på tidlige integration mellem alle store komponenter for at sikre, at alt kommunikation mellem frontend, game engine, backend og databasen har fungeret fra et tidligt tidspunkt i projektet. I stedet for at integrere og teste alting samtidigt, er hvert komponent blevet gradvist integreret i projektet. Det har vist sig nemmere at integrerer mange små ændringer end at lave en stor integration. 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 (REF HERE).

Den stores success rate skyldes en insistens på tidlige integration mellem alle store komponenter for at sikre at alt kommunikation mellem frontend, game engine, backend og databasen har fungeret fra et tidligt tidspunkt i projektet. I stedet for at integrere og teste alting samtidigt, er hvert komponent blevet gradvist integreret i projektet. Det har vist sig nemmere at integrerer mange små ændringer, end at lave en stor integration til sidst i udviklingsfasen.



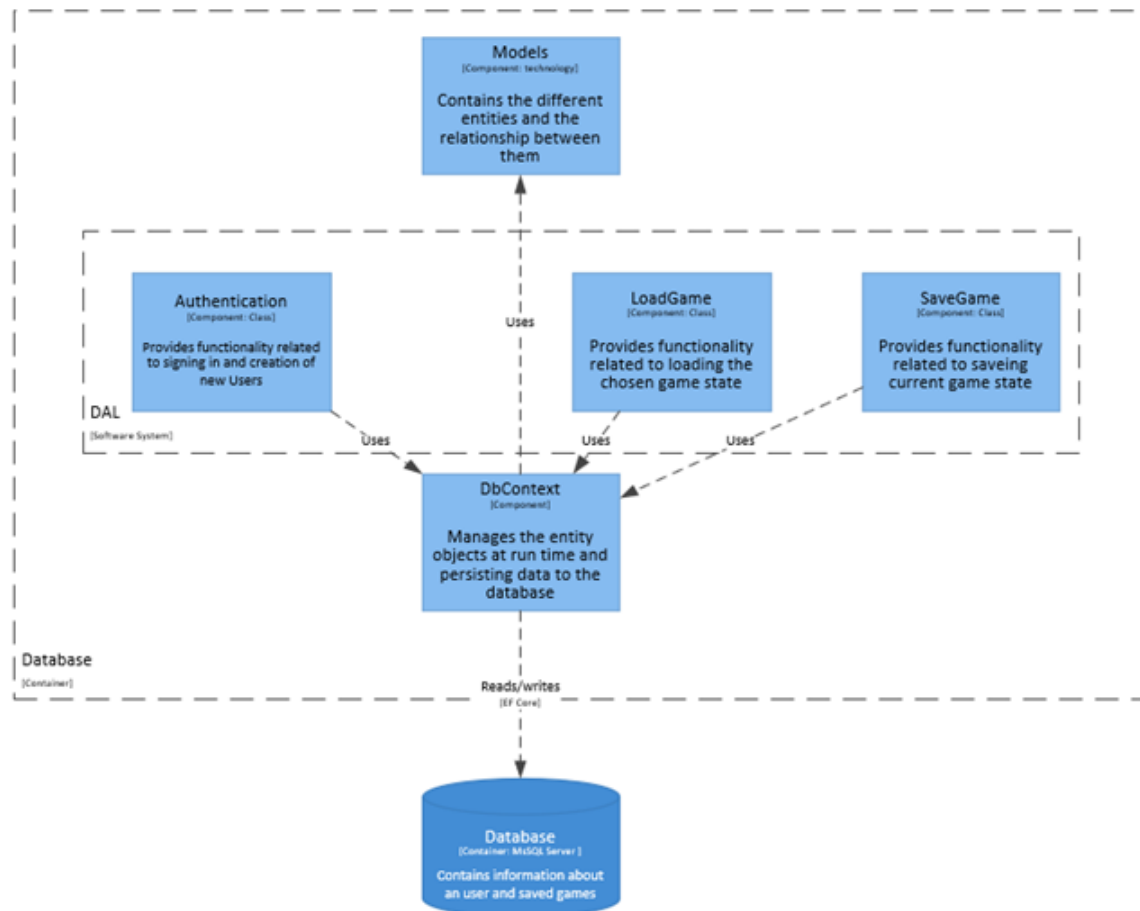


Figure 16: C3 diagram for blokkene involveret i DAL's funktionalitet

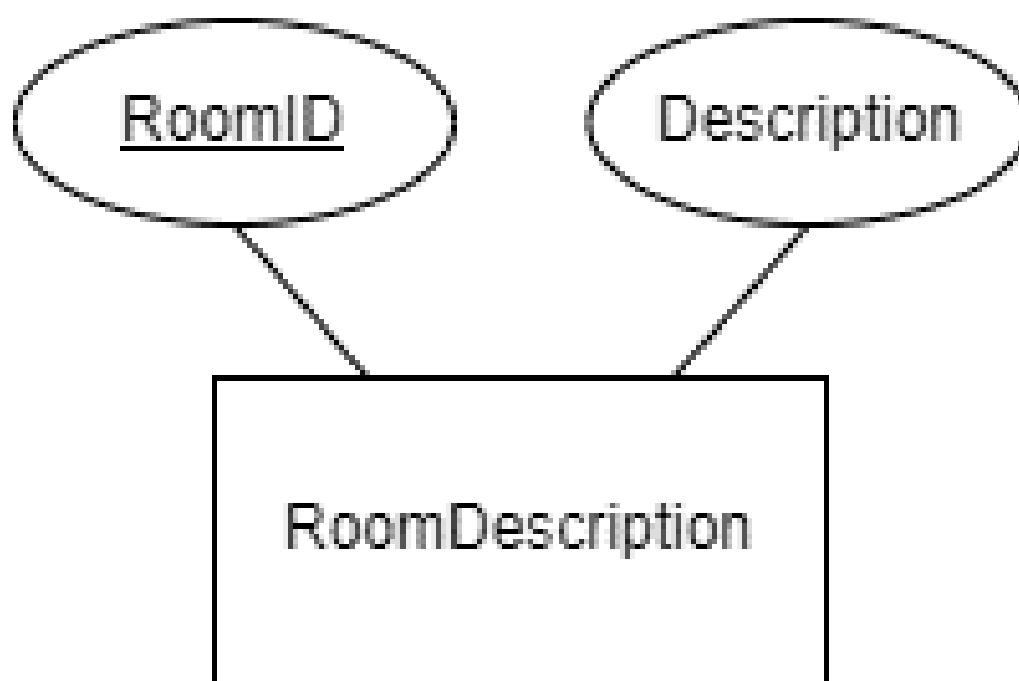


Figure 17: ER diagram for Roomdescription. En beskrivelse består blot af en beskrivende string samt det tilhørende unikke rumid.

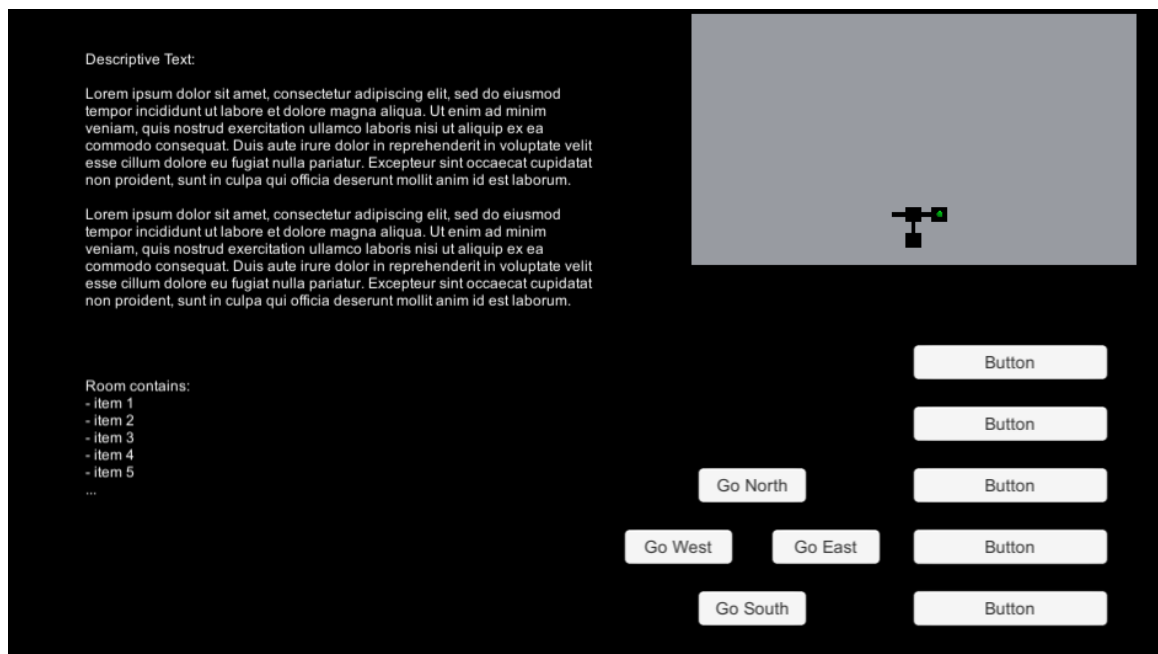


Figure 18: Et mockup af det primære spil vindue. Tekst øverst i venstre side af skærmen giver en beskrivelse af det rum spilleren er i, samt en liste af elementer i rummet som spilleren kan interagere med. Øverst til højre vises et billede af banen. Spilleren interagerer med spillet via knapper nederst 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 funktionaliteter i spillet.

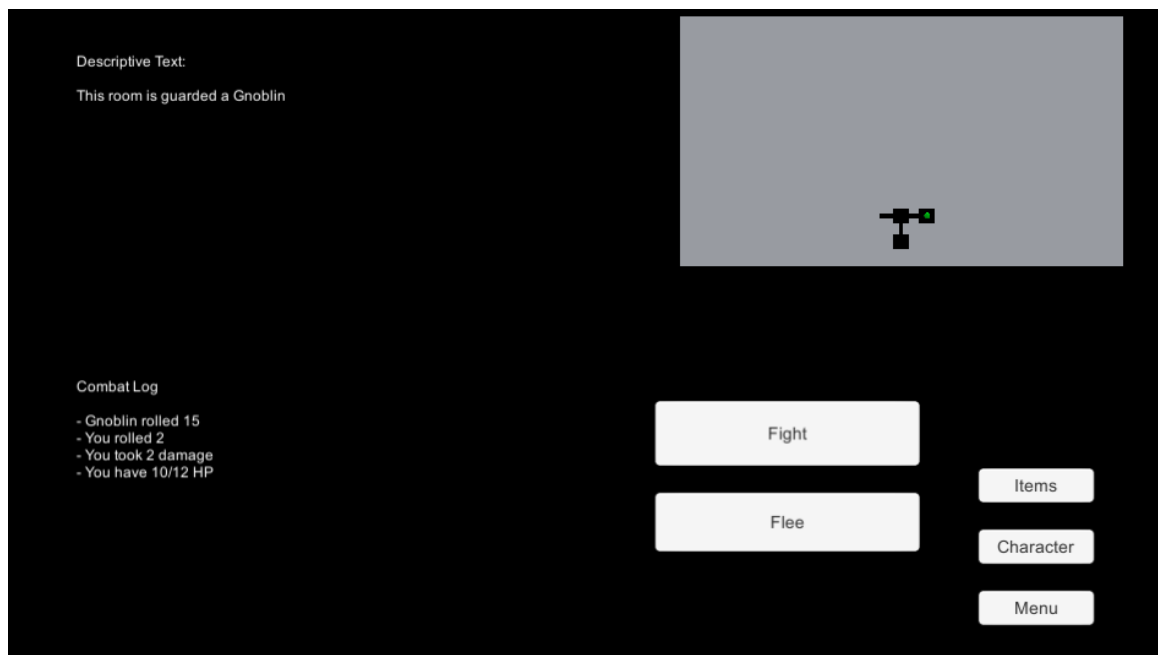

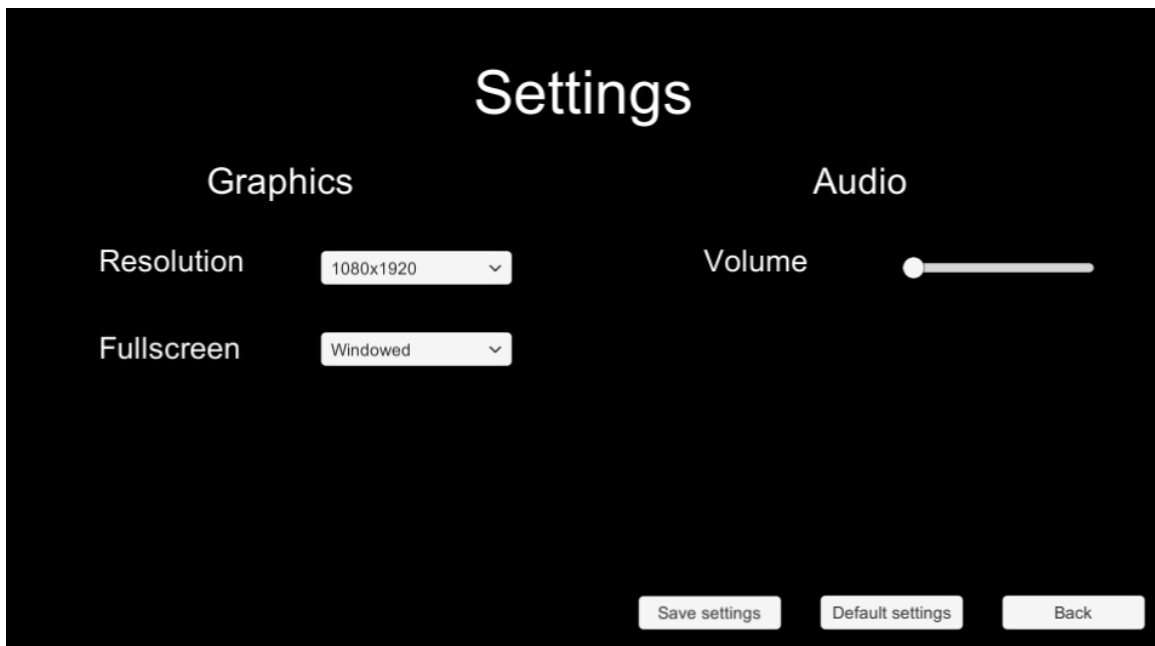


Figure 19: Combat view. Spilleren præsenteres for en fjende, og får information om hvordan kampen mod fjenden går. Der er knapper til at kæmpe og flygte, samt gå til menuer. Øverst til højre er kortet over banen, ligesom i Room vinduet Figure 18.



The screenshot shows a login interface with a dark background. At the top, the word "Title" is displayed in a large, white, sans-serif font. Below it, the text "Username:" is followed by a white text input field containing the placeholder text "Enter text...". This is followed by the text "Password:" and another white text input field with the same placeholder. Below the password field are three white buttons stacked vertically: "Login", "Create User", and "Exit".

Figure 20: Login view. Bruger kan indtaste sit brugernavn og kodeord for at få adgang til spillet, eller trykke på create user for at komme til et vindue hvor man kan oprette en ny bruger. Det er også muligt at forlade spillet igen.



The screenshot shows a settings interface with a dark background. At the top, the word "Settings" is displayed in a large, white, sans-serif font. Below it, the interface is divided into two columns. The left column is titled "Graphics" and contains two settings: "Resolution" with a dropdown menu showing "1080x1920" and a downward arrow, and "Fullscreen" with a dropdown menu showing "Windowed" and a downward arrow. The right column is titled "Audio" and contains a "Volume" slider with a white knob and a horizontal track. At the bottom of the screen are three white buttons: "Save settings", "Default settings", and "Back".

Figure 21: Settings view. Tillader at man kan ændre indstillinger for spillet, så som skærmopløsning og lydstyrke. Det er muligt at gemme indstillingerne, forlade skærmen uden at gemme indstillingerne, samt sætte spillet tilbage til standard indstillinger.

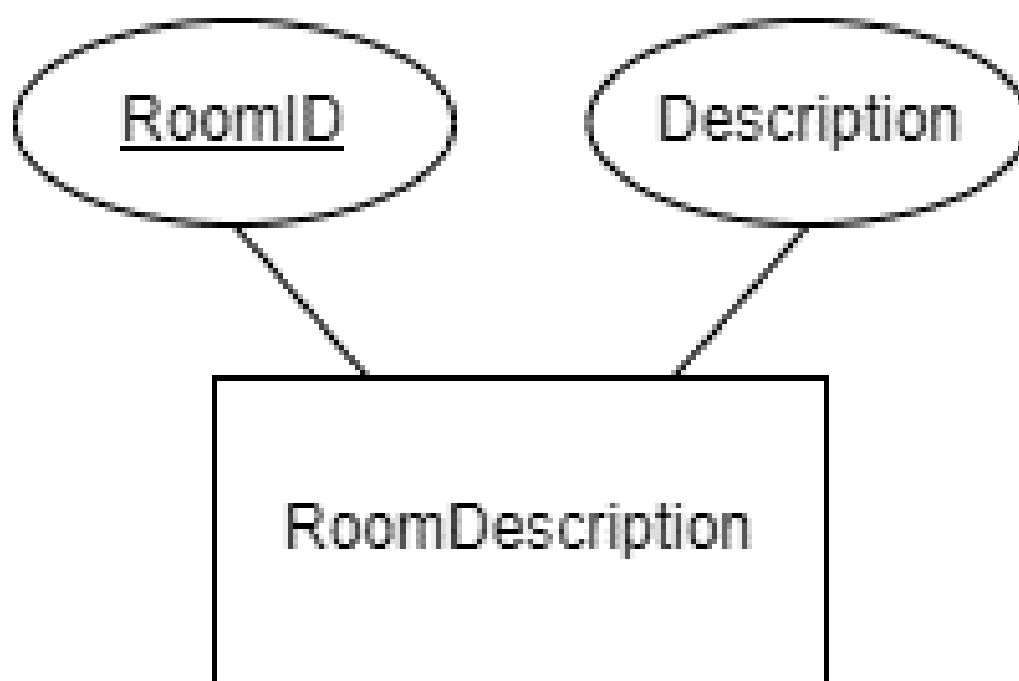


Figure 22: ER diagram for Roomdescription. En beskrivelse består blot af en beskrivende string samt det tilhørende unikke rumid.



Figure 23: ER Diagram til at gemme et spil til en specifik bruger

The login screen for the game "Dungeons & Goblins" features a black background with white text and input fields. At the top, the title "Dungeons & Goblins" is displayed. Below it, the label "Username" is followed by a text input field. Further down, the label "Password" is followed by another text input field. At the bottom, there are two buttons: "Login" and "Register". In the bottom right corner, there is a link labeled "Exit Game".

Figure 24: Endelig login skærm. Brugenavn og kodeord kan indtastes i de to felter. Register knappen fører til et nyt view, hvor man kan oprette en bruger, mens login knappen fører spilleren til main menu, hvis deres login er korrekt.



Figure 25: 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.

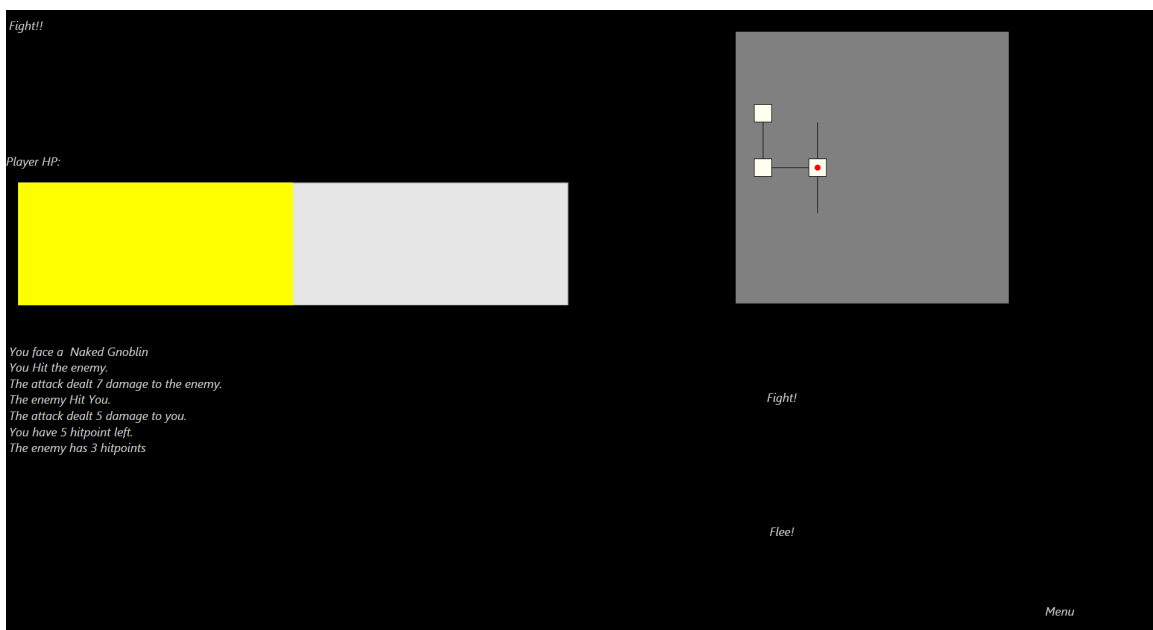


Figure 26: Combat view er baseret på room view. Her præsenteres spilleren for info om en fjende og hvordan kampen mod fjenden går.

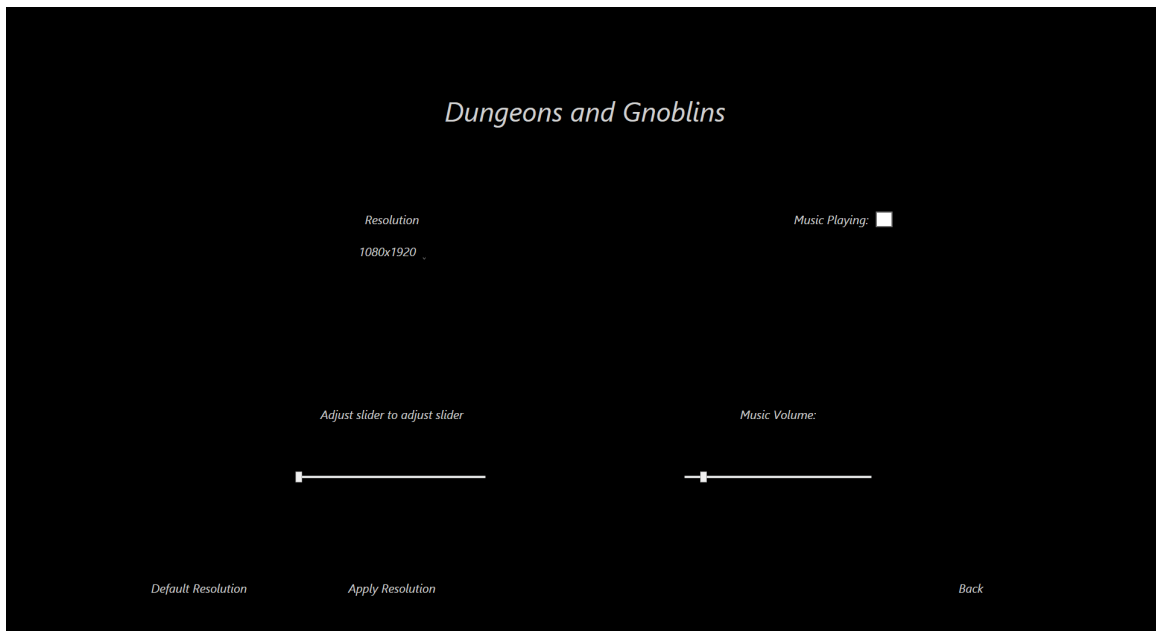


Figure 27: Menu til at ændre spillets indstillinger. Det er her muligt at vælge skærmopløsning og lydstyrke, samt tænde og slukke for musikken. Back knappen fører tilbage til enten main menu eller ingame menu, afhængig af hvilke menu man tilgik settings menuen fra.

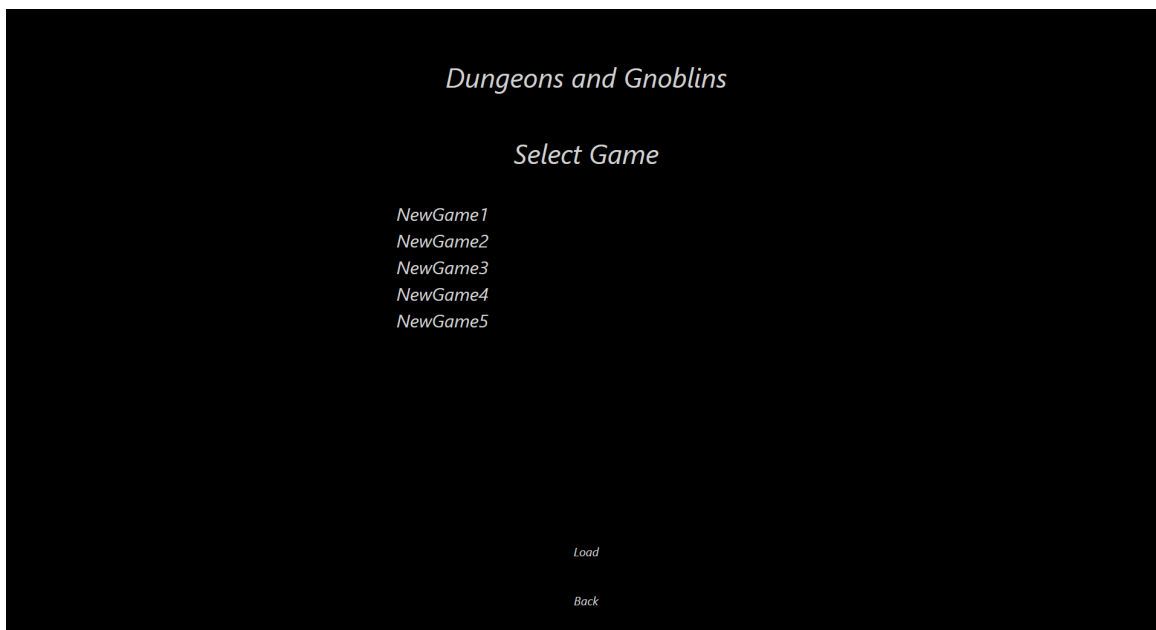
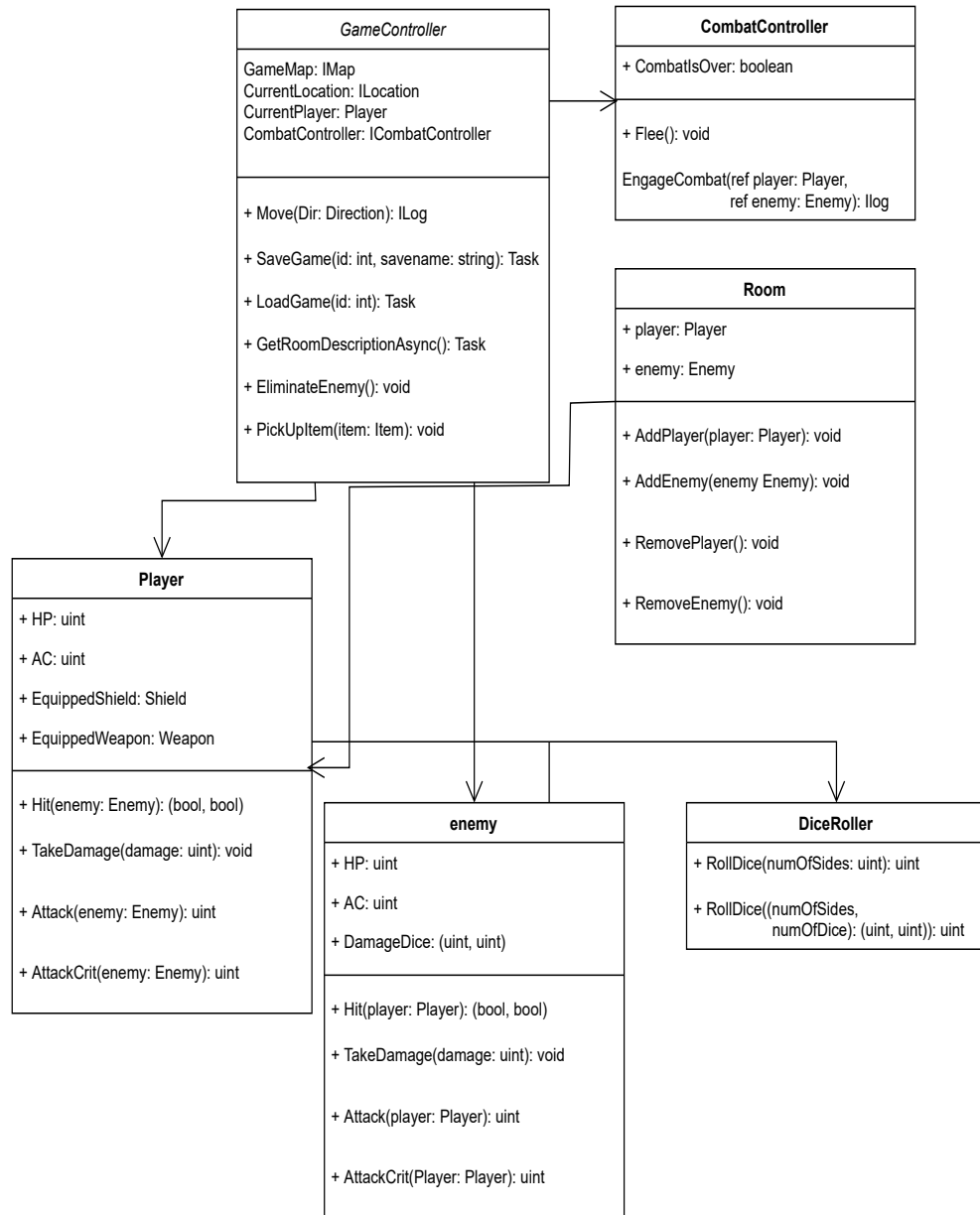


Figure 28: Load menu view. Her præsenteres spilleren for alle gemte spil. Disse hentes fra databasen hver gang spilleren går ind i load menuen.

Figu
Der

nden.

Figure 30: Load funktionen, resetter game, setter de rigtige states og gennemløber alle Room instances og sætter deres state til den korrekte state.

```
public async Task LoadGame(int id)
{
    await Reset();
    SaveDTO Game = await backEndController.GetSaveAsync(id);
    CurrentLocation.RemovePlayer();
    CurrentLocation = GameMap.Rooms[Game.RoomId];
    VisitedRooms = Game.VisitedRooms;
    SlainEnemies = Game.SlainEnemies;
    Inventory = Game.Inventory;
    CurrentPlayer.HP = Game.Health;
    GameMap.Rooms[CurrentLocation.Id].AddPlayer(CurrentPlayer);
    List<(uint, Item)> temparray = new List<(uint, Item)>();

    foreach (ILocation room in GameMap.Rooms)
    {
        if (room.Chest != null)
        {
            foreach (Item item in room.Chest)
            {
                if (Inventory.Contains(item.Id))
                {
                    if (Game.WeaponId == item.Id)
                    {
                        CurrentPlayer.EquippedWeapon = (Weapon)item;
                    }
                    if (Game.ShieldId == item.Id)
                    {
                        CurrentPlayer.EquippedShield = (Shield)item;
                    }
                    CurrentPlayer.Inventory.Add(item);
                    temparray.Add((room.Id, item));
                }
            }
        }
    }

    foreach ((uint TempRoom, Item TempItem) tempval in temparray)
    {
        GameMap.Rooms[tempval.TempRoom].Chest.Remove(tempval.TempItem);
    }

    if (room.Enemy != null)
    {
        if (SlainEnemies.Contains(room.Enemy.Id))
        {
            room.RemoveEnemy();
        }
    }
}
```

Figure 31: viser illustrer hvordan en linje fra map layout filen omdannes til en liste af room id'er som rummet er forbundet med. Dette er kerne mekanismen for hvordan en spiller kan navigere rundt i spillet, da en spiller ikke kan bevæge sig fra et Room til et anden, der ikke er i denne liste af forbindelser mellem det nuværende room og den ønskede bevægelses retning.

```
1 reference
private LinkedList<int> CreateRoomLink(string linkingString)
{
    int[] link = linkingString.Split(",").Select(int.Parse).ToArray();
    return new LinkedList<int>(link);
}
```

Figure 32

```
[TestFixture]
0 references
public class PlayerTest
{
    private Player uut;
    private Enemy enemy;
    private DiceRoller basicDiceRoller;
    private Weapon weapon;

    [SetUp]
    0 references
    public void SetUp()
    {
        basicDiceRoller = Substitute.For<DiceRoller>();
        weapon = Substitute.For<Sword>(((uint) 8, (uint) 1), (uint) 2, (uint) 2);
        uut = new Player(10, 16, weapon, basicDiceRoller);
        enemy = new Enemy(10, 10, (10, 1), 2, "test");
    }
}
```

Figure 33: Mocks benyttes her til at sikre at dependencien, her DiceRoller, returner den ønskede værdi for situationen, der er under test. Alle test tildeles informative navne, for at sikre læsbarhed i forhold til testenens formål.

```
[Test]
0 references
public void Hit_IfPlayerEquippedWeaponNull_DefaultHitIsFalse()
{
    basicDiceRoller.RollDice(20).Returns((uint)10000);
    uut.EquippedWeapon = null;
    bool expectedHitResult = false;
    (bool Hit, bool) actualHitResult = uut.Hit(enemy);
    Assert.That(actualHitResult.Hit, Is.EqualTo(expectedHitResult));
}
```

Figure 34: Alle skrevne 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.

