Gnomes and Gnoblins

Semesterprojekt

Aarhus Institue of Technology

Author:
Morten Høgsberg, 201704542

Date: May 24, 2022

Contents

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.....(diskuteres) Rum på mappet loades efethånden som man besøger dem.

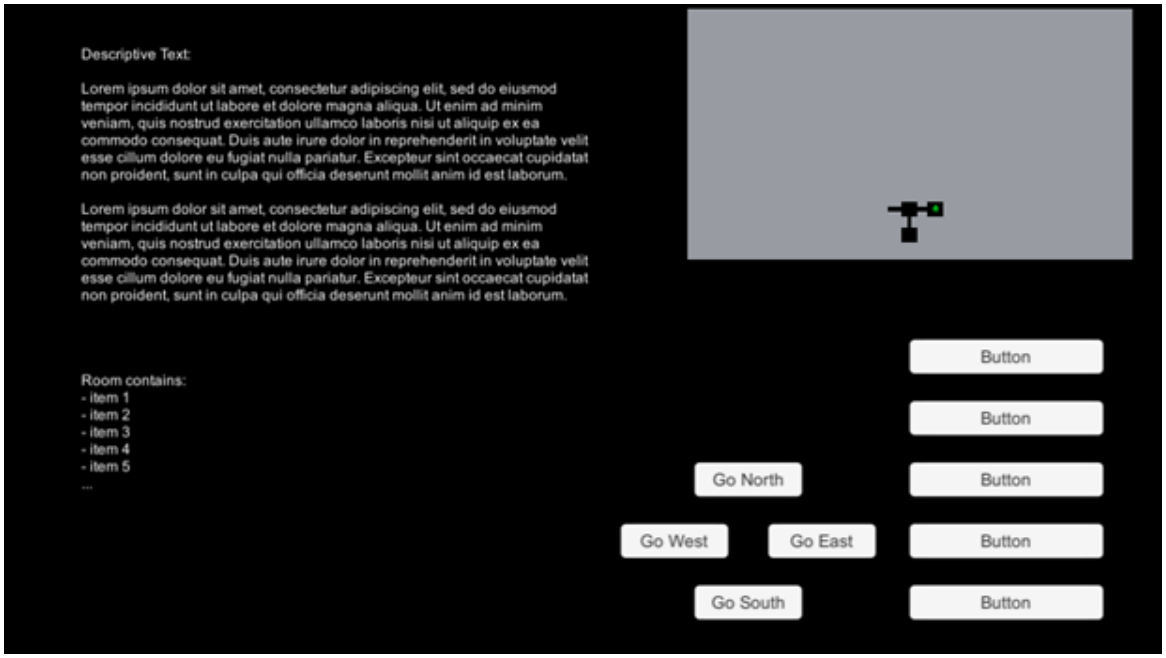

Figure 2

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.

Arkitektur

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

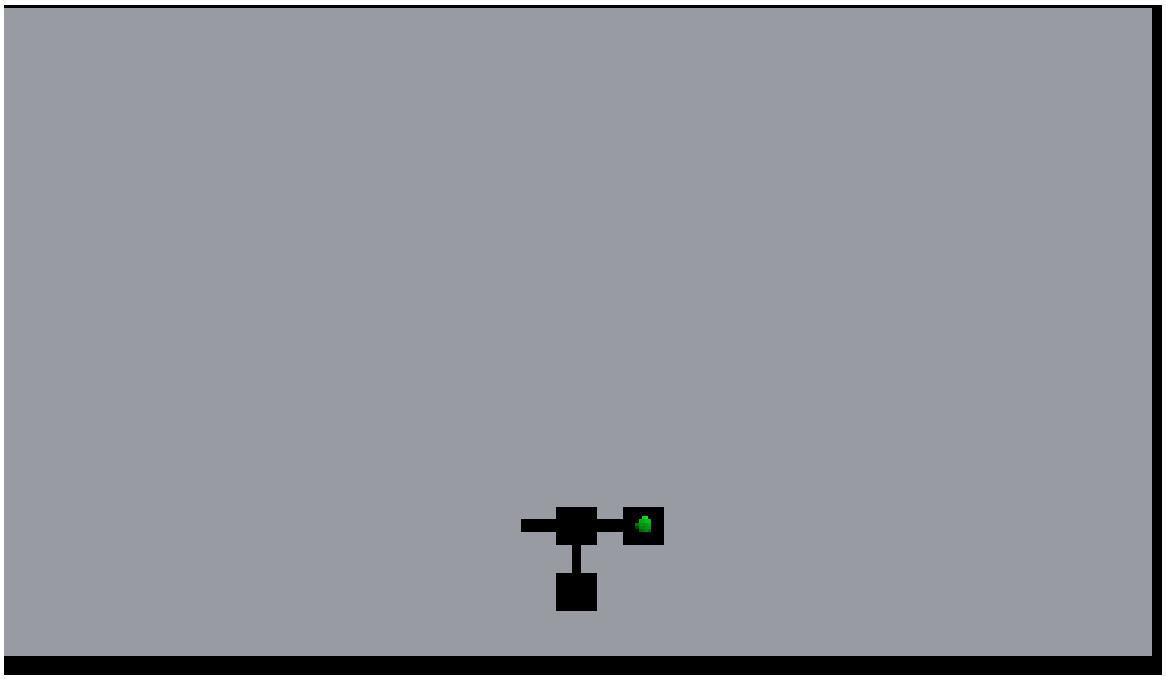

Figure 3: Closeup Screenshot af mappet

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 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. 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

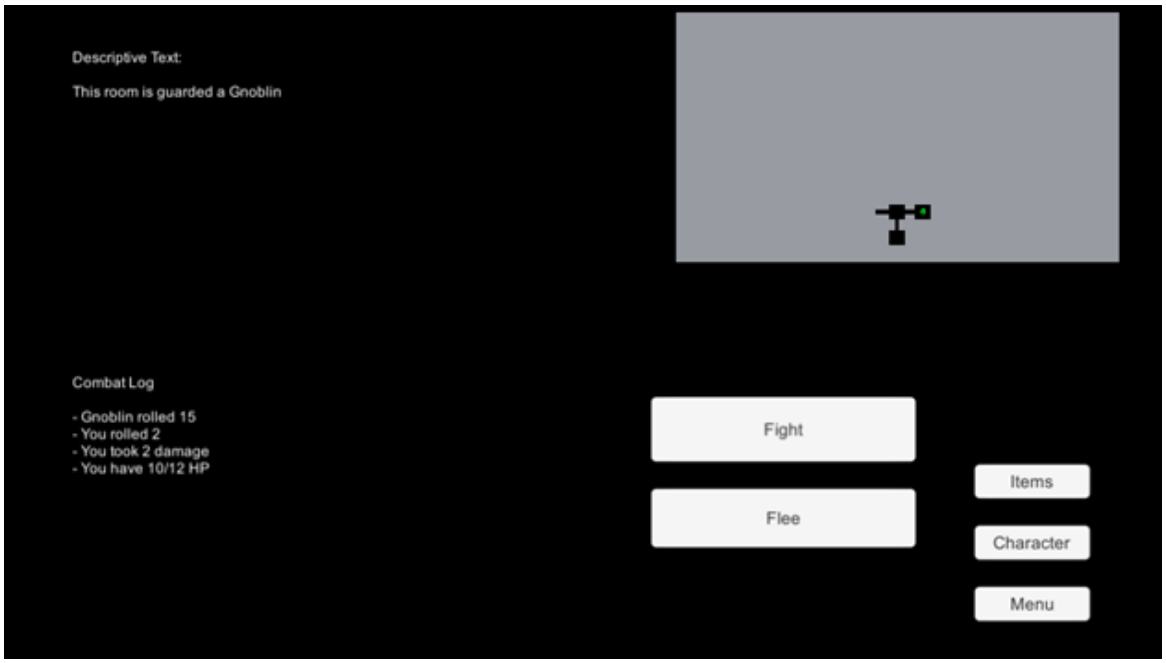

Figure 4

- Register
- Save Game
- Load Game

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

Tilsidst 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 userstory minder på mange måder om "Save Game", men i stedet for at sende et element til databasen, skal der hentes et gemt spil fra databasen, med alle de data der skal bruges for at kunne loade sit spil op præcis som man forlod det.

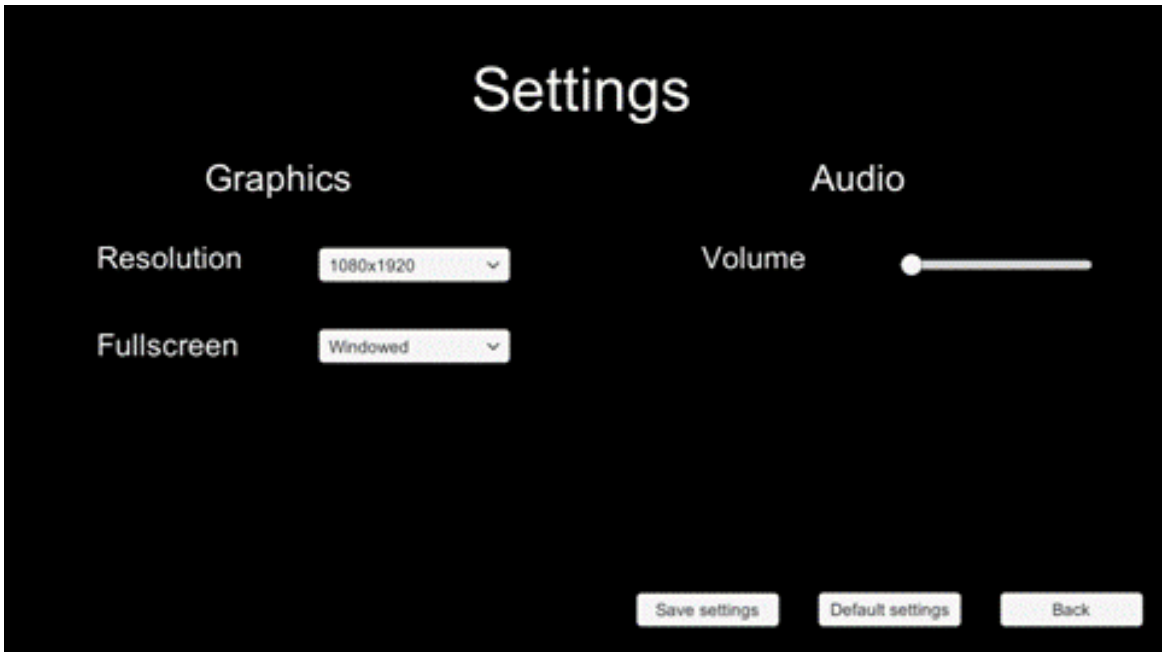

Figure 5: Screenshot af settings menuen

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 2.4. Spillet vil bestå af en række af vinduer, som giver spilleren den nødvændige 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 udviklingsværktøh 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.
Her følger en række af mockups[1] af nogle af spillets views.

Room

Room view (Figure 11) 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 forbingelse 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.

[1] Lavet i Unity

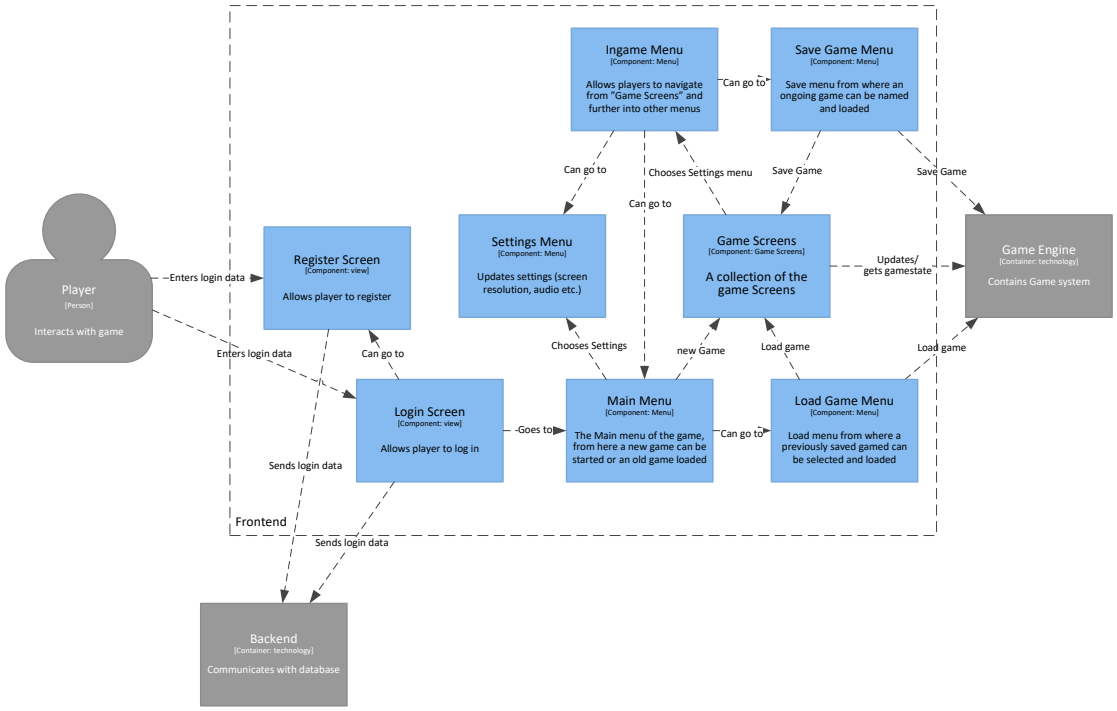

Figure 6: C3-Model for Frontend. Modellen fortæller hvordan man kan navigere igennem forskellige menuer og hvilke menuer der kan føre til hvad. Derudover kan man se hvilke blokke der snakker ud af frontenden og sammen med resten af systemet.

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 oprætte en ny bruger.
Exit lukker spillet.

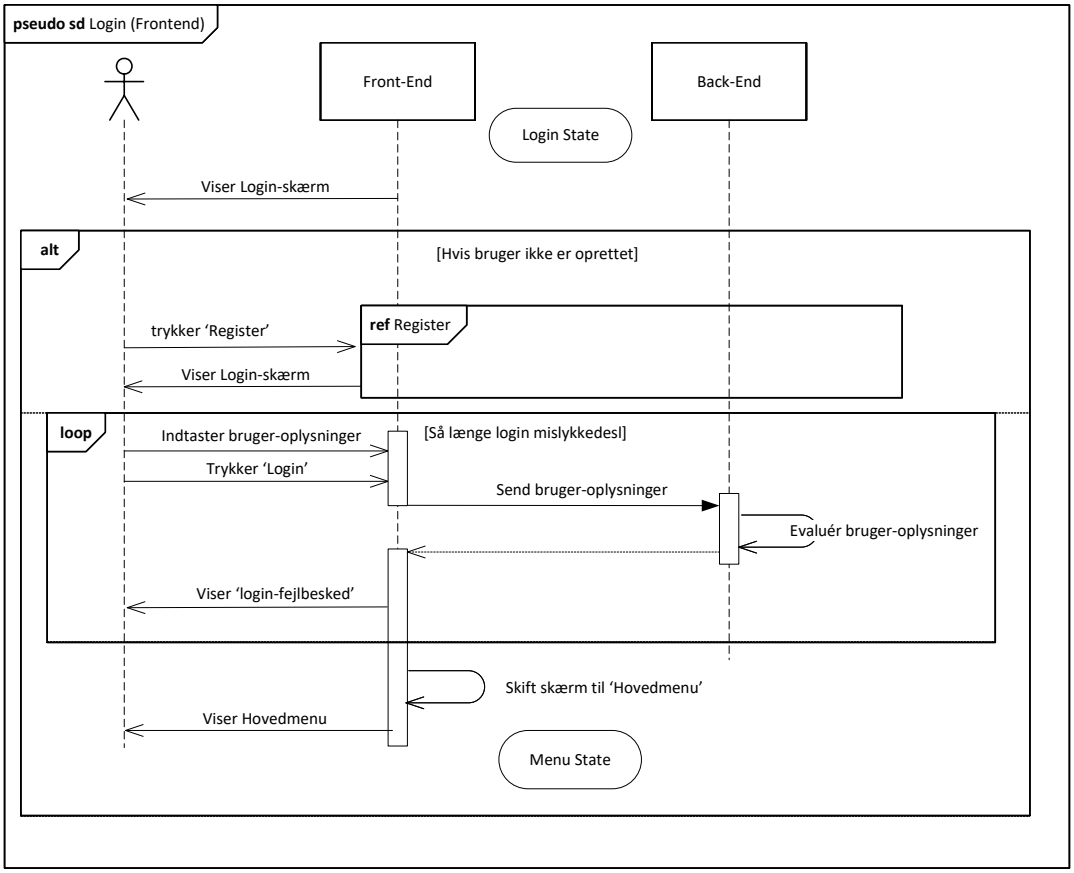

Figure 7: Pseudo sekvensdiagram af forløbet af userstory "Login", set fra Frontends perspektiv. Med reference til "Register" userstory og håndtering af forkerte login oplysninger.

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.

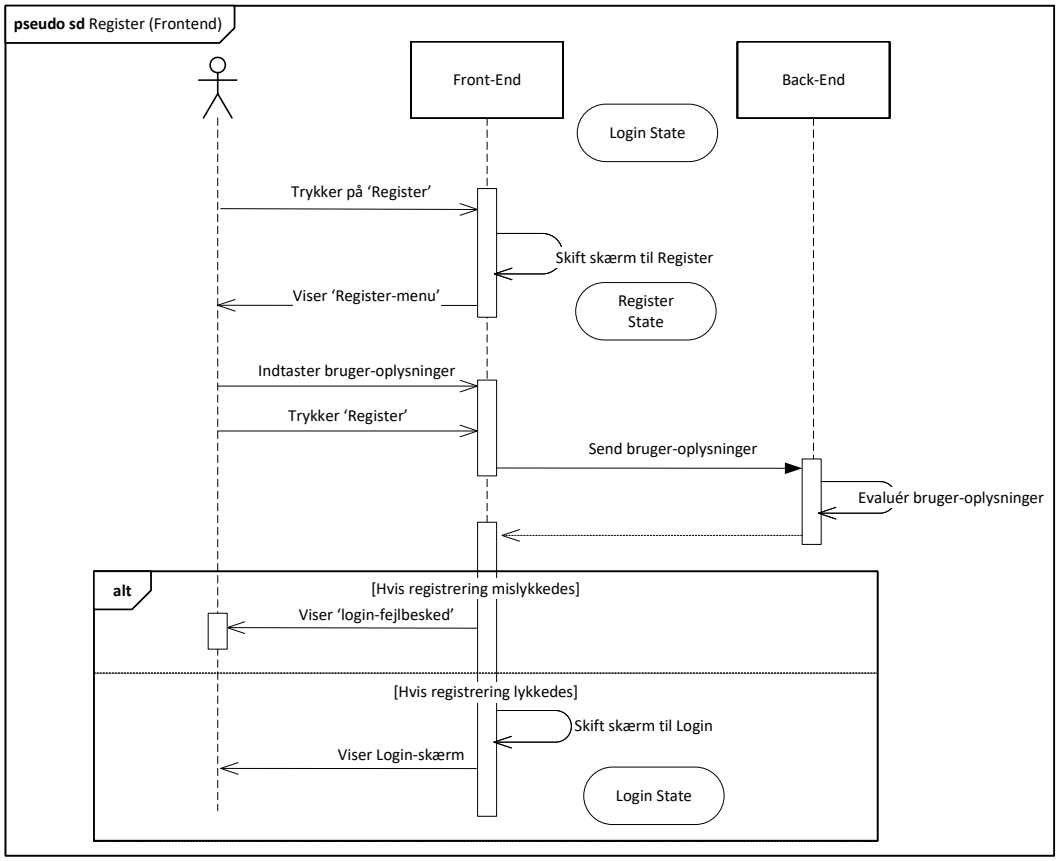

Figure 8: Pseudo sekvensdiagram af forløbet af userstory "Register", set fra Frontends perspektiv. Med håndtering af forkerte login oplysninger.

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 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.

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 ønkede vi at gemme en række forskellige attributter for spilleren. Første og

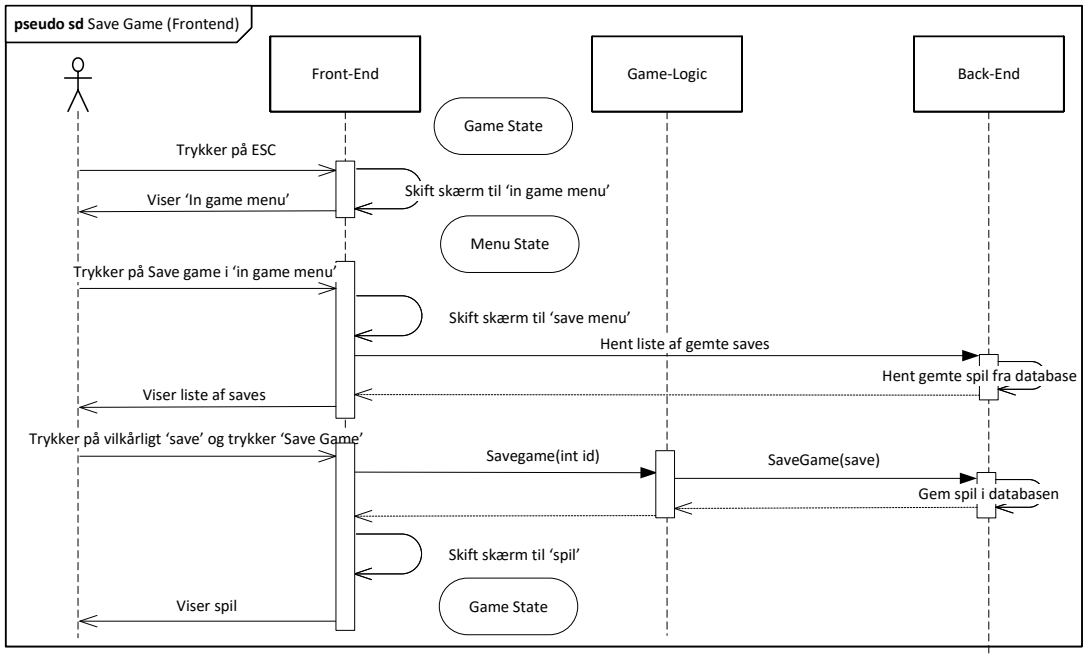

Figure 9: Pseudo sekvensdiagram af forløbet af userstory "Save Game", set fra Frontends perspektiv. Der laves 2 kald til databasen igennem Backenden, hvori der i det første kald, "Hent liste af gemte saves" hentes en liste af brugerens gemte spil og i andet kald gemmes brugerens nuværende spil henover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

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 enimy 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.

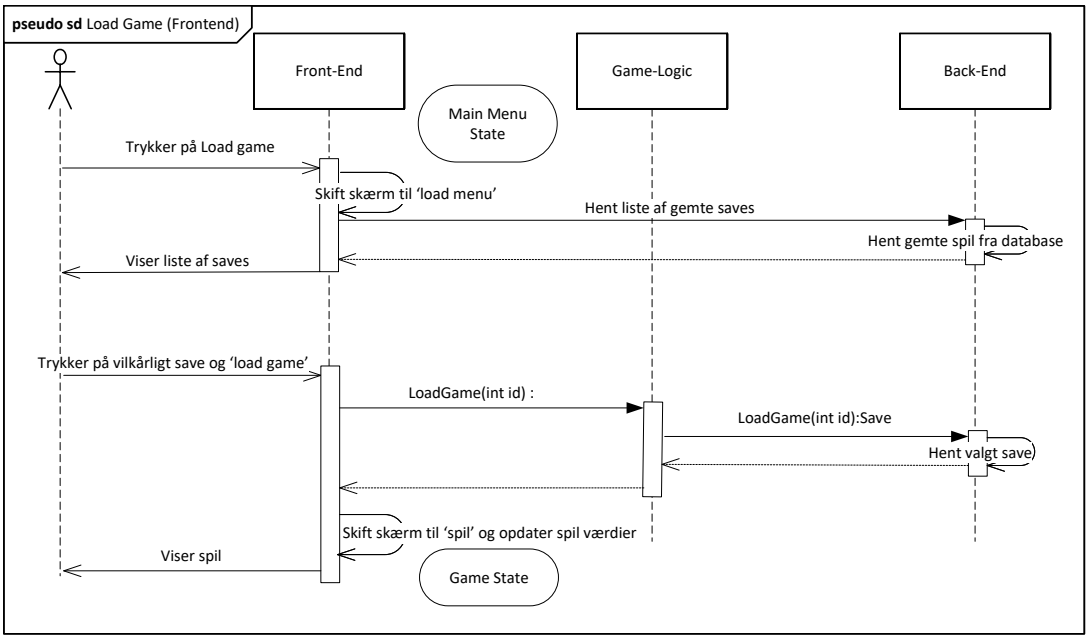

Figure 10: Pseudo sekvensdiagram af forløbet af userstory "Load Game", set fra Frontends perspektiv. Der laves 2 kald til databasen igennem Backenden, hvori der i det første kald, "Hent liste af gemte saves" hentes en liste af brugerens gemte spil og i andet kald hentes brugerens valgte spil og spillet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

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 derved 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 nødvændigt 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 interesante programerings-tekniske beslutninger.

Login view

De overordnede struktur af login (Figure 17) 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 resourses og styles i app.xaml filen(**INDSÆT REFERENCE HER**). Dette gør det nemt

[2] **https://www.technical-recipes.com/2018/navigating-between-views-in-wpf-mvvm/**.

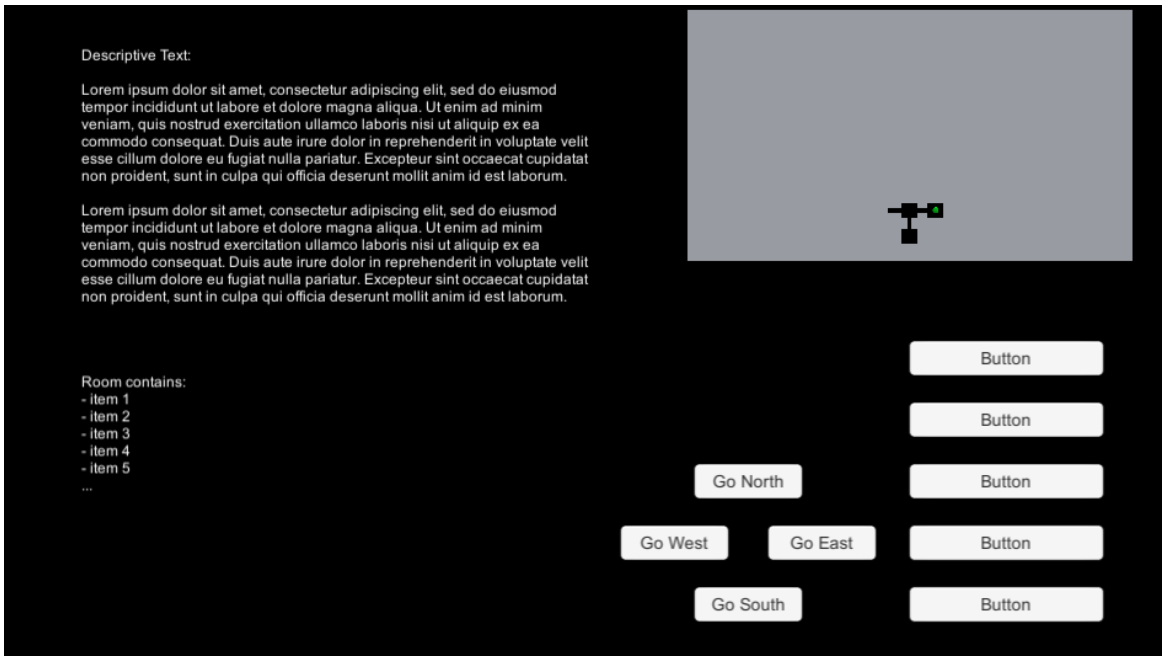

Figure 11: Et mockup af det primære spil vindue. Tekst øverst i venstre side af skærmen giver en beskrivelse af det rum spilleren er i, samt en liste af elementer i rummet som spilleren kan interagere med. Øverst til højre vises et billede af banen. Spilleren interagerer med spillet via knapper nederst 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.

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 18) 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 understøtes.
Ved at trykke på interact knappen kan spilleren flytte et valgt 'item' fra listennederst til venstre over i sit inventory (et seperat view), som kan tilgås veed at trykke på Inventory knappen.
Alt tekst er vist med data binding.

Combat View

Combat view (Figure 19) 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

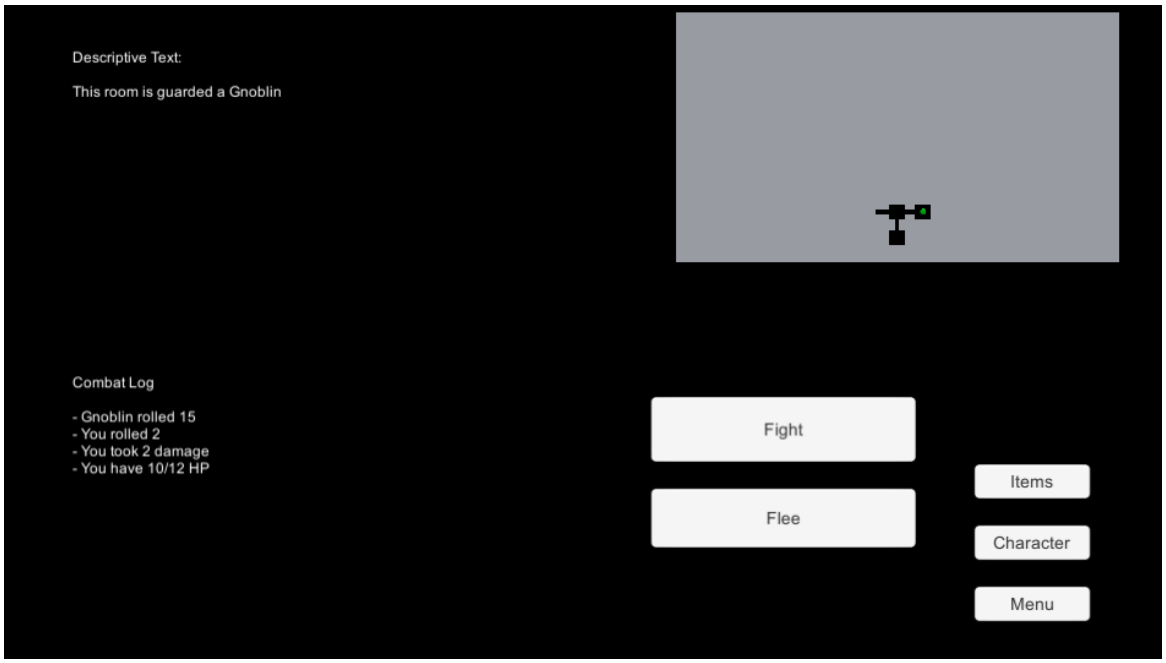

Figure 12: Combat view. Spilleren præsenteres for en fjende, og får information om hvordan kampen mod fjenden går. Der er knapper til at kæmpe og flygte, samt gå til menuer. øverst til højre er kortet over banen, ligesom i Room vinduet Figure 11.

spillet og skifter frave 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 20) 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 læselige. Kortet i Room og Combat view skalerer med skærmopløsningen, men sætter en nedre grænse på 300x300 pixel. Tekststørelsen 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

Figure 13: 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 oprætte en ny bruger. Det er også muligt at forlade spillet igen.

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 21) 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 templaten (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**

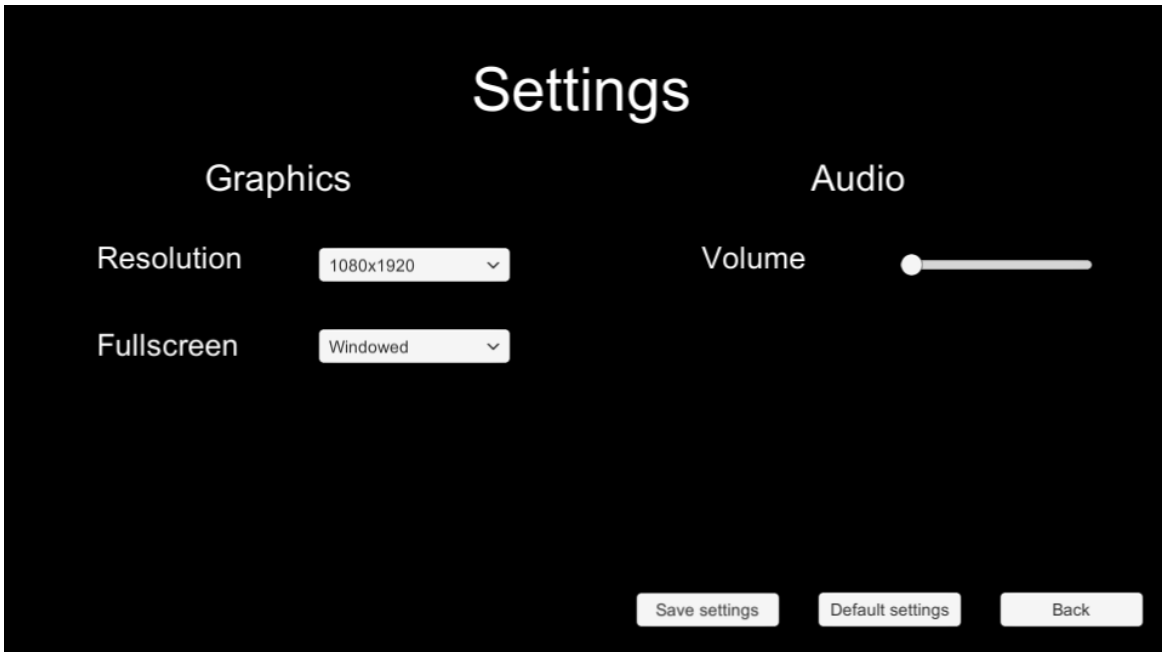

Figure 14: Settings view. Tillader at man kan ændre indstillinger for spillet, så som skærmopløsning og lydstyrke. Det er muligt at gemme indstillingerne, forlade skærmen uden at gemme indstillingerne, samt sætte spillet tilbage til standard indstilinger.

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 inkludere automatiseret unittests, integrationstest og accepttest.

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 førger til at Game Engine er eksklusivt test ved hjælp af black-box testing. Alle testene tester kun det offentlige interface, som er gjort tilgængelig.

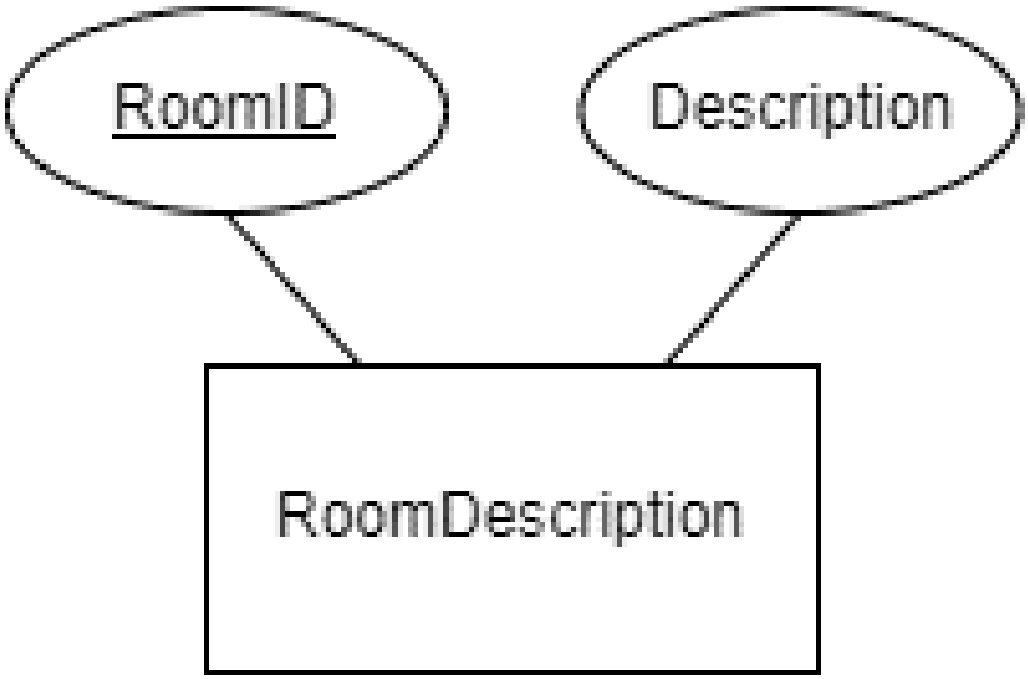

Figure 15: ER diagram for Roomdescription. En beskrivelse består blot af en beskrivende string samt det tilhørende unikke rumid.

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 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 Har kun test for at skifte til nyt Room	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 engager i combat.	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.	OK
BaseMapCreator	1. Kan generere et map layout file	BaseMapCreator can på korrekt vis generere et map layout file	OK
BaseMap	1. Kan anskaffe Rooms ud fra en Direction	Map kan anskaffe Rooms uf fra en given Direction.	OK

Player	1. Kan angribe Enemy 2. Kan tage skade 3. Kan skade Enemy (Ikke Kristisk) 4. Kan skade Enemy (Kritisk)	Player kan angribe, skade og tage skade fra enemy.	OK
Enemy	1. Kan angribe Player 2. Kan tage skade 3. Kan skade Player (Ikke Kristisk) 4. Kan skade Player (Kritisk)	Enemy kan angribe, skade og tage skade fra Player.	OK
Log	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	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 construeres på korrekt vis.	OK

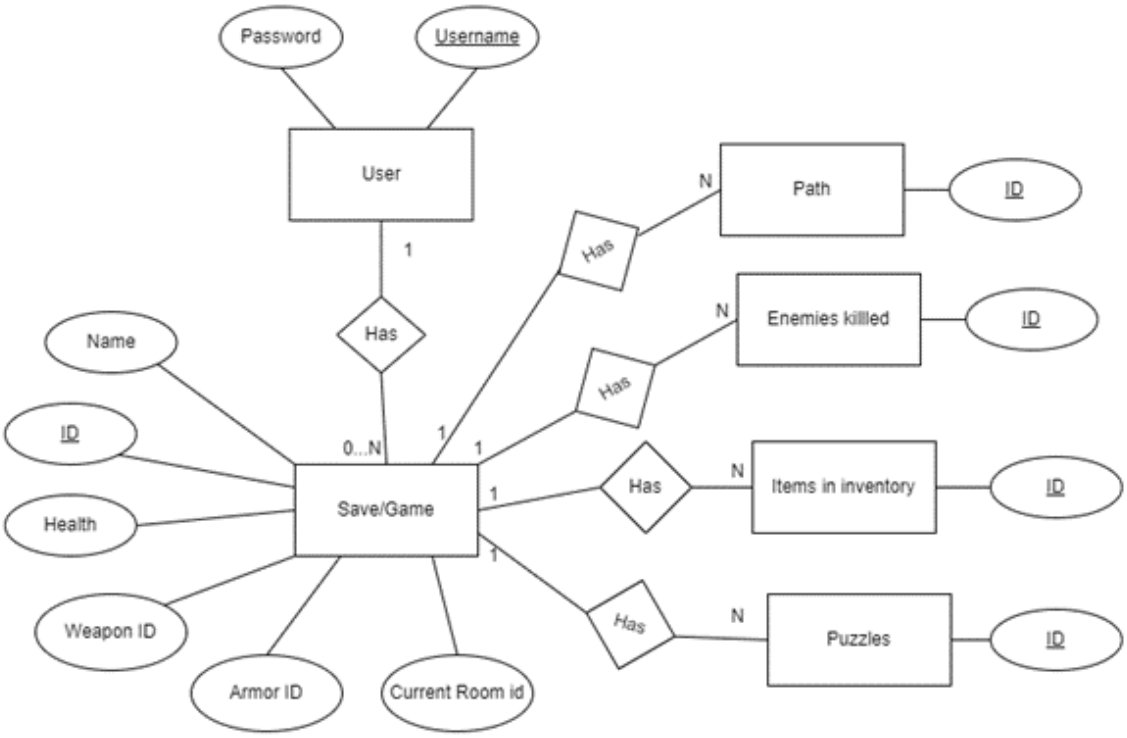

Figure 16: ER Diagram til at gemme et spil til en specifik bruger

Mock testing

I nogle tilfælde kan det være umuligt er 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 senarier for Player klassen testes, hvor man kan regne med at DiceRoller giver et bestemt output.

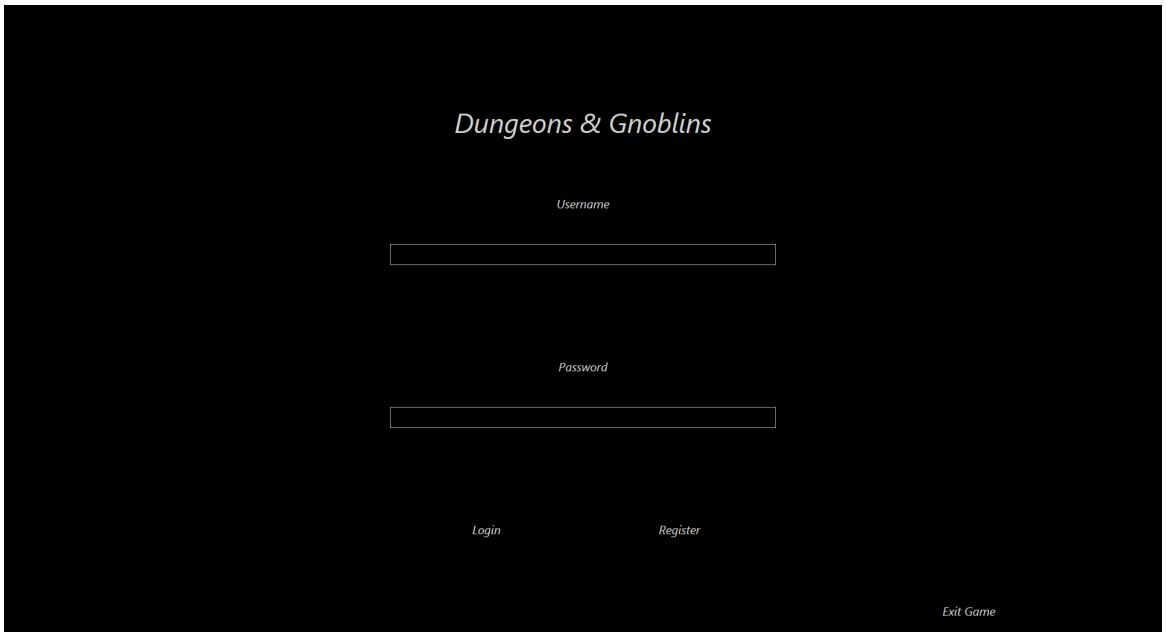

Figure 17: Endelig login skærm. Brugenavn og kodeord kan indtastes i de to felter. Register knappen fører til et nyt view, hvor man kan oprætte en bruger, mens login knappen fører spilleren til main menu, hvis deres login er korrekt.

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.

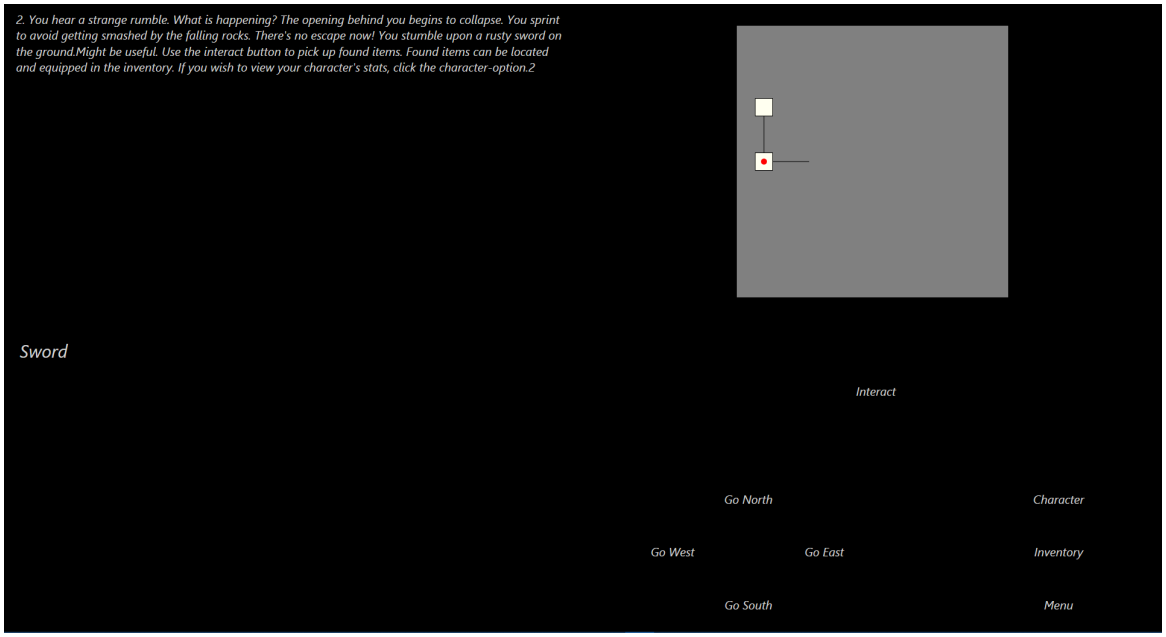

Figure 18: Endeligt udseende af room view. Generelt er der ikke ændret meget i forhold til det oprindelige design. Kortet er lavet så det skalerer med skærmopløsningen.

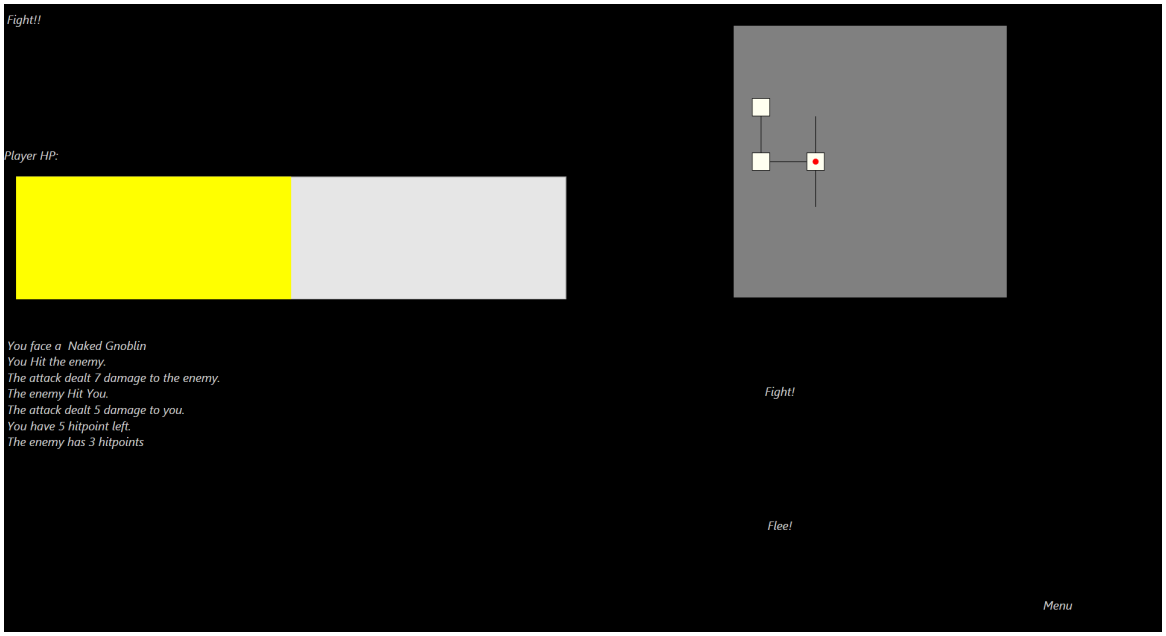

Figure 19: Combat view er baseret på room view. Her præsenters spilleren for info om en fjende og hvordan kampen mod fjenden går.

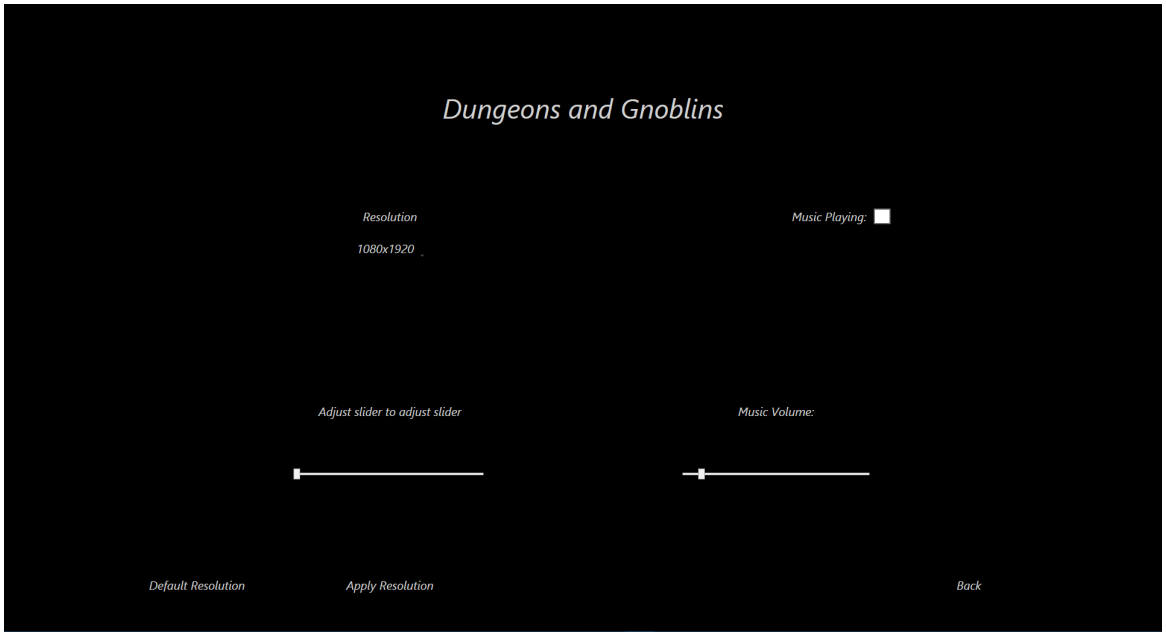

Figure 20: Menu til at ændre spillets indstillinger. Det er her muligt at vælge skærmopløsning og lydstyrke, samt tænde og slukke for musikken. Back knappen fører tilbage til enten main menu eller ingame menu, afhængig af hvilke menu man tilgik settings menuen fra.

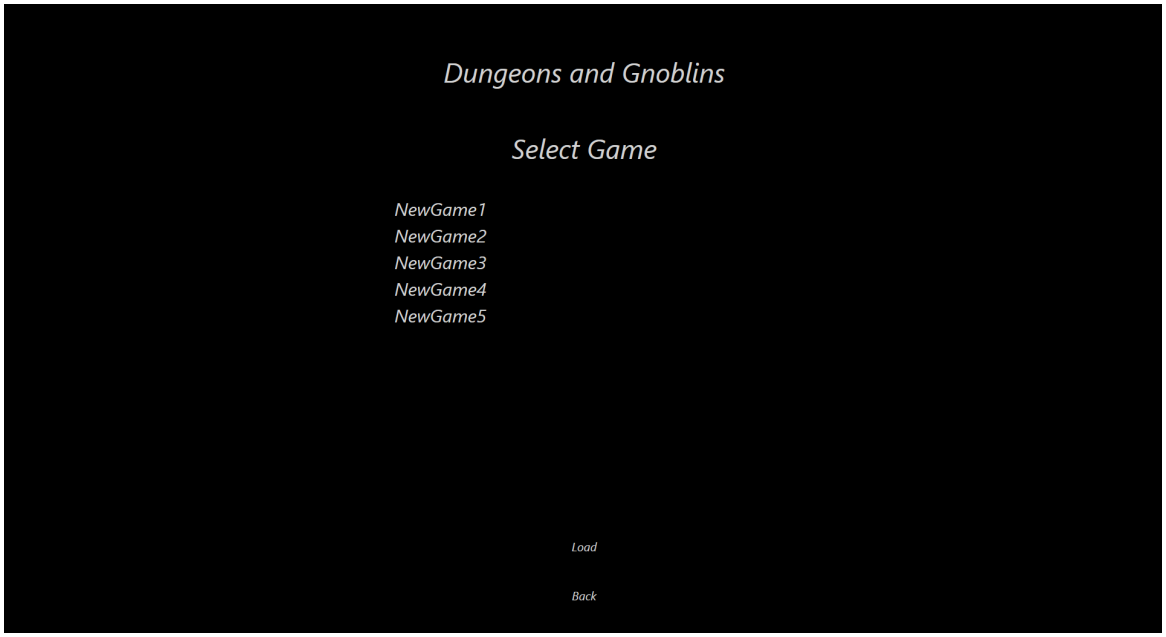

Figure 21: Load menu view. Her præsenteres spilleren for alle gemte spil. Disse hentes fra databasen hver gang spilleren går ind i load menuen.

Figure 22

```
[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 23: 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 testenes 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 24: 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.