

# Projektuppgift DT071G

*DT071G*

**Konsol Spel**

Enkelt Text Baserat RPG

**Max Gagner (maga2101)**

**MITTUNIVERSITETET**  
**Institutionen för data- och elektroteknik(DET)**

**Författare:** Max Gagner, [maga2101@student.miun.se](mailto:maga2101@student.miun.se)

**Utbildningsprogram:** Webbutveckling, 120 hp

**Huvudområde:** Datateknik

**Termin, år:** 01, 2024

## **Sammanfattning**

I denna projektrapport så detaljeras frågeställningen, utföringen och resultatet utav ett konsol baserat spel byggt i C# .NET. Spelet ska byggas med någon form utav användargränssnitt, samt olika tekniker ifrån tidigare i kursen. Dessa tekniker är främst implementationen och träningen utav machine learning models, och läsningen/skrivningen utav JSON data för att hantera information.

# Innehållsförteckning

Sammanfattning .....	iii
1 Terminologi och Teori.....	v
Akronymer/Förkortningar .....	v
1.1 Machine Learning Modell .....	v
1.2 Threading.....	v
2 Introduktion .....	1
2.1 Bakgrund och problemmotivering .....	1
2.2 Avgränsningar .....	1
2.3 Detaljerad problemformulering.....	2
3 Metod .....	3
4 Konstruktion .....	4
4.1 UI .....	4
4.1.1 Row-Print .....	4
4.1.2 Slow-Print.....	4
4.1.3 Continue-Dotter .....	6
4.2 Meny och fillertext.....	7
4.2.1 Text.....	7
4.3 Meny.....	7
4.4 Karaktär och Spelnivå Generation.....	8
4.5 MLM.....	9
4.6 Spel, Sparning och Combat.....	9
5 Resultat.....	11
6 Slutsatser .....	12

# 1 Terminologi och Teori

För förståelse utav projektrapporten och projektet i sig så krävs förkunskaper inom grundläggande C# eller liknande programmeringsspråk såsom C++. Utöver detta så är kunskap inom machine learning modells (MLM) och threading fördelaktigt, men inte nödvändigt för förståelse av texten.

## **Akronymer/Förkortningar**

MLM	Machine Learning Modell
-----	-------------------------

### **1.1 Machine Learning Modell**

Machine Learning Modells inom detta kontext är ett "objekt" som är tränat för att känna igen och identifiera mönster, i syftet att kategorisera eller framställa resultat baserat på indata. Till exempel så kan den vara tränad att se skillnaden på "Nej" och "Ja" med värdet utvärdet 1 och 2 respektive till de olika textstyckena. En MLM kräver träningsdata för att framställa resultat, varav större en mängd data gör MLM'en mer precisare.<sup>1</sup>

Inom detta projekt så används MLM för att analysera användarinputs och kategorisera dessa med ett värde för att utföra operationer baserat på detta värde.

### **1.2 Threading**

I C# .NET så finns det funktionalitet runt att utföra operationer parallellt som heter multithreading, Denna funktionalitet låter utvecklaren att skapa funktioner som excaveras parallellt med den huvudsakliga processen för att få asynkrona resultat. Helt enkelt så låter det utvecklaren skapa threads som kan utföra sysslor parallellt.<sup>2</sup>

<sup>1</sup> <https://learn.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model>

<sup>2</sup> <https://learn.microsoft.com/en-us/dotnet/standard/threading/using-threads-and-threading>

## 2 Introduktion

### 2.1 *Bakgrund och problemmotivering*

Inom detta projekt så fick utföraren själv välja ämnet och metodiken för att skapa en valfri typ utav applikation. Med det enda utsatta målet var att inkludera tidigare metodik som till exempel JSON manipulation och sparning. Program ska dessutom vara interaktivt med någon form utav gränssnitt för användaren att använda.

De delar utav kursen som valdes att fokusera på var hanteringen av data genom JSON och användandet utav MLM för att tolka användarinput. Dessa valdes eftersom utföraren ansåg att dessa var de intressantaste och bredaste delarna utav de tidigare kurs-uppgifterna.

Efter fokuset hade framställts så valdes att utveckla ett konsol-baserat "rollspel" spel som fungerar genom att användaren matar in vad dem vill att spelkaraktären utför och en MLM tolkar detta som en vald gärning. Målet med spelet kommer att vara att besegra fiender i en nivå-baserad miljö med högre svårighetsgrad desto högre nivå användaren når.

Detta passar bra för att använda de båda metodikerna i stor utbredning- Då MLM kan användas för att tolka all indata och JSON kan användas för lagring och sparning utav speldata.

### 2.2 *Avgränsningar*

Ett fullständigt spel som kan anses som "komplett" kommer inte att skapas under utvecklingsprocessen. Främst på grund av antalet text och bredden utav innehåll som behöver skapas för detta syfte. Men all grundläggande funktionalitet för spelets funktion kommer att finnas.

### **2.3 Detaljerad problemformulering**

Det finns ett antal utmaningar som medförs på grund utav att projektet är ett spel. En stor del av dessa utmaningar härstammar i att spelet använder konsolen som ett grafiskt medium. Vilket ärver en stor del restriktioner och nackdelar som kommer med den grundläggande konsolen. Såsom omöjligheten att stoppa användare ifrån att skriva i mitten utav en process, och limiterade grafiska funktionaliteter.

Utöver UI (användargränssnitt) så är det även spelets innehåll som kan bli problematiskt, text och fiende-design kan ta en stor mängd tid att utveckla. Framför allt i ett större spel där en viss nivå av variation krävs för att erhålla kontinuerligt intresse. Däremot så är detta projekt avgränsat inom just detta fält vilket minskar magnituden av detta hinder men stoppar det inte helt.

MLM är en av huvudfunktionaliteterna som kommer att tolka användarinput för att derivra numeriska utfall baserat på tanken bakom användarens text. Detta kräver en del testning och träning med MLM samt skapandet av omfattande träningsdata för att minska potentiella felaktiga utfall. Största problemet här är skapandet och formaterandet av träningsdata, eftersom det inte finns någon större informationsbank som kan användas för detta syfte.

Slutligen så är själva struktureringen utav projektet i sig en utmaning, hur ska sparning gå till? Ska det finnas ett slut? Hur vet olika delar av koden den överhängande statusen?

Alla problem behöver lösas för att uppnå en välformad lyckad produkt.

### **3 Metod**

Till projektet så kommer Microsoft Visual Studio<sup>3</sup> användas för alla kod relaterade syften (2022 versionen). För versionshantering så kommer GIT<sup>4</sup> att användas. Dessutom så används Chatbotten ChatGPT-4o-mini<sup>5</sup> under några delar av projektet för att generera större summor JSON data i tids-sparnings syfte.

<sup>3</sup> <https://visualstudio.microsoft.com/>

<sup>4</sup> <https://git-scm.com/>

<sup>5</sup> <https://chatgpt.com/>



## 4 Konstruktion

### 4.1 UI

Första problemet med projektets utveckling var att få standard konsolen att vara mer grafiskt "uttrycksfull", till exempel utforska möjligheterna kring enkla animationer o.s.v. Det första som behövdes tacklas var hur text visades, när text skrivs i konsolen så skriver den direkt allt som ska skrivas i ett block. Vilket är en bra och effektiv metod, men inte i detta fall då ett spel blir mer intressant om den har en "unik" grafisk stil.

För att göra textutskrifter mer dynamiskt och intressant så skapades en klass med tre olika funktioner för omfattande texthantering. Som sedan kan användas för att hantera och presentera praktiskt taget all textinformation genom konsolen.

#### 4.1.1 Row-Print

Första funktionen som skapas till utskrifts-syfte är Row-Print, denna funktion skriver ut en skickad string med en bas paus på 100 millisekunder efteråt. Denna funktion skapas för att göra en "animation" för att rendera textstycken från toppen till botten med en paus mellan varje stycke. Den ger texten en "fallande" animation.

#### 4.1.2 Slow-Print

Den andra funktionen skapades för att skriva ut texten symbol för symbol, helt enkelt emulera hur manuell läsning antas se ut. Detta löses genom att använda strings, på grund av att strings är en char array så går det helt enkelt att loopa igenom arrayen och skriva en outprint för varje symbol.

Men detta löste inte helt problemet, då en char inte är särskilt mycket information så skrivs det ut i sådan hastighet att det inte är en märkbar skillnad ifrån att skriva block för block. Detta löses i sin tur genom att lägga till en manuell paus efter varje symbol.

Men konsol texten är fortfarande inte "intressant" nog, det finns ingen funktionalitet för att separera viktig information ifrån brödtext. Därför skapas det möjligheten att skicka med en string med färgnamn för att färglägga specifika delar utav texten när den skrivs ut.

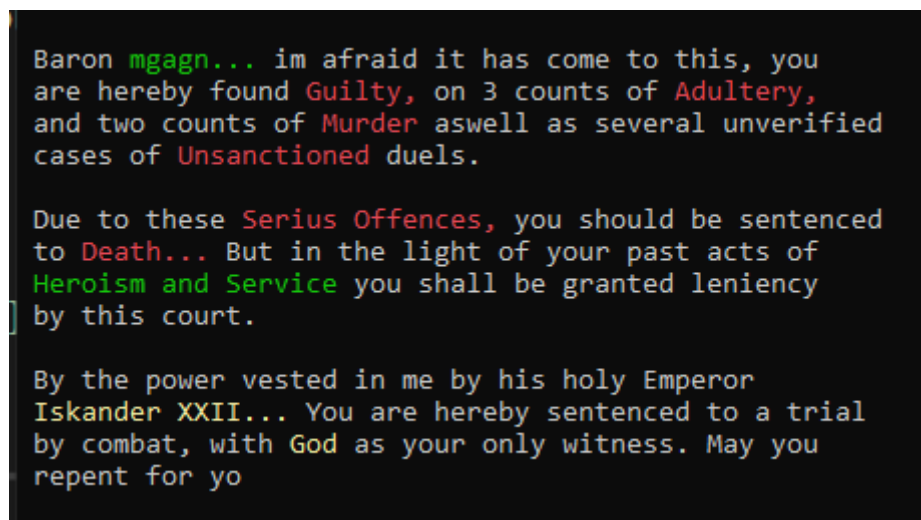
Detta leder till att texten kan skrivas ut symbol för symbol med möjligheten att färglägga stycken efter behov. Med den enda större nackdelen att det är väldigt ineffektivt, men detta är inte ett riktigt problem då det är ett spel och inte en applikation som ska köra 100 instanser parallellt.

Två problem uppstod under utvecklingen av denna funktion, första är att textstyckena i konsolen kan vara extremt breda då de baseras efter konsol storleken. Det andra är att punkter och kommas inte respekterades, de skrivs ut som vilka andra symboler som helst. Vilket fungerar, men det fyll-

ler inte den grafiska visionen för konsolen Därav så fick punkter och kommas en längre paus genom en enkel if sats, vilket ger en illusion av pauser inom läsning.

Styckes längden är däremot problematiskt. Därav så skapas en överhängande variabel som håller koll på text längden och en annan som bestämmer den största styckes längden. Varefter programmet skriver till max-längden och när den är uppfylld så kollar den efter nästa mellanrum för att bryta rad.

Sista delen utav funktionen är att inkorporera globala variabler ifrån en "settings" klass för att dynamiskt kunna ändra hastigheten av texten och maximala text-längden. Detta ger en användare mer konsol och gör spelet mer tillgängligt.



```
Baron mgagn... im afraid it has come to this, you
are hereby found Guilty, on 3 counts of Adultery,
and two counts of Murder aswell as several unverified
cases of Unsanctioned duels.

Due to these Serious Offences, you should be sentenced
to Death... But in the light of your past acts of
Heroism and Service you shall be granted leniency
by this court.

By the power vested in me by his holy Emperor
Iskander XXII... You are hereby sentenced to a trial
by combat, with God as your only witness. May you
repent for yo
```

Figur 1 Textexempel tagit i mitten av utskrift

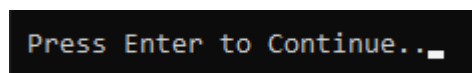
### 4.1.3 Continue-Dotter

Denna konstigt nämnda funktionen skapas för att försöka göra en grafisk animation eller uppdatering inom restriktionerna utav konsolen. Det är en funktion som byggs för att köra en annan thread som agerar parallellt med huvudprocessen. För att stoppa spelets text ifrån att vara ett massivt block i konsolen så delas den upp i olika skärmar eller "nivåer". När en utav dessa en är slut så rensas själva konsolen.

Men ett problem skapas utav detta. Vad händer om användaren inte har läst klart? Detta problem blev anledningen till denna funktion.

Funktionen är relativt simpel, den skriver ut en punkt och sedan väntar 2 sekunder innan den skriver en till. När den har kört 3 gånger så använder den "backspace" tre gånger för att ta bort punkterna och börja om.

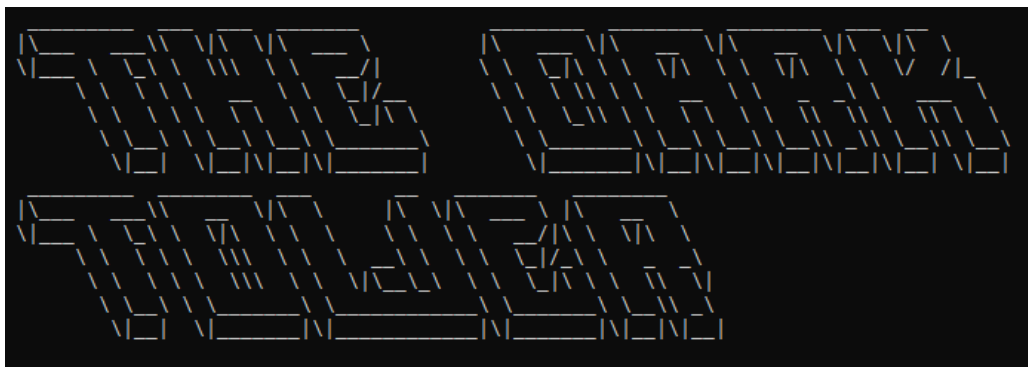
Detta kombinerat med en "press x to continue" prompt och en funktion som väntar på önskat input skapar en bra lösning till detta problem. Varefter den funktionen får det önskat inputen så stoppar den threaden och fortsätter vidare.



Figur 2 Paus som väntar på "enter" innan den går vidare.

Den sista delen utav den grafiska designen av spelet, är skapandet utav en bra "titel". Med det menas det att ha någon form utav bild eller liknande "konstverk" för att förmedla titeln och fånga användarens intresse.

Däremot att skriva ut en bild i konsolen är komplicerat till omöjligt, men det går att använda ASCII "art" för att skapa pseudo-bilder. Därefter så användes webbplatsen ASCII Art Archive<sup>6</sup> för att konvertera texten "The Dark Tower" till en passande ASCII "art" som sedan kan skrivas ut genom row-print för att skapa en intressant "start-animation".



Figur 3 Spel Titeln i konsolen

<sup>6</sup> <https://www.asciiart.eu/text-to-ascii-art>

## 4.2 Meny och fillertext

### 4.2.1 Text

Efter de grafiska funktionerna har utvecklats så behövs stycken utav text skapas för att fylla spelet med brödtext. Varefter dessa textstycken delas upp i strings och sedan köras med de olika textfunktionerna.

Eftersom detta projekt enbart är ett prototypspel och inte ett fullständigt, så skapas bara en del fillertext med målet att skapa en nivå av "atmosfär" till spelet. För att nå en bra nivå utav inlevelse behövs en bakomliggande story till spelet finnas och texten skapas för att uppnå detta.

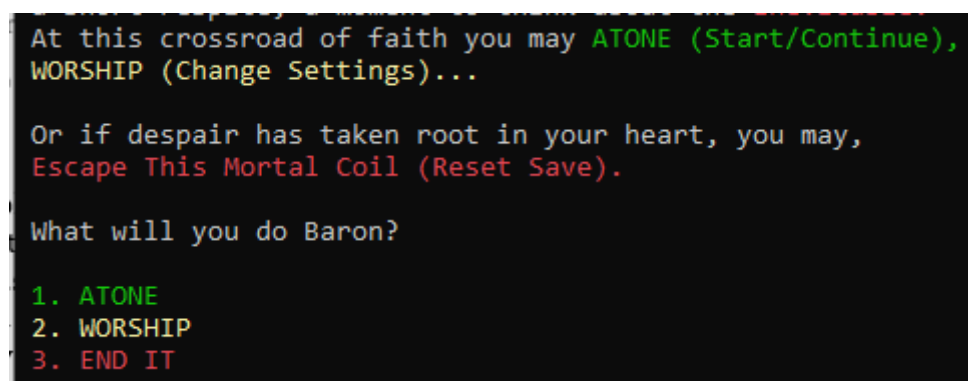
För att bibehålla användarens intresse inom de större text-styckena så används färgläggning utav ord med uppfattad vikt. I slutliga målet att skapa mer "intressant" och lättläslig text.

## 4.3 Meny

Spelet ska innehålla funktionalitet kring sparning och sparfiler, vilket kräver något interface där användaren kan utföra dessa gärningar. Till detta skapas en meny som öppnas vid starten utav spelet. I menyn så kan användaren utföra olika operationer som till exempel starta spelet.

Men det uppkom även ett behov att kunna ändra spelets inställningar som till exempel den tidigare nämnda tecken-hastigheten. Varefter det skapas funktionalitet för att öppna en annan "sida" där det går att ändra på texthastigheten och liknande inställningar. Däremot så skapar detta en situation där meny texten skrevs ut först när användaren kom in i menyn, och igen när hen kom tillbaka.

Detta är problematiskt då upprepad text bara slösar användarens tid. Därför skapas en array utav bools som kollar om användaren nyss har varit i inställningar eller menyn. Därefter ökas läkningstiden utav respektive sida till praktiskt taget omedelbar för att inte få användaren att läsa samma text igen.



```
At this crossroad of faith you may ATONE (Start/Continue),  
WORSHIP (Change Settings)...  
  
Or if despair has taken root in your heart, you may,  
Escape This Mortal Coil (Reset Save).  
  
What will you do Baron?  
  
1. ATONE  
2. WORSHIP  
3. END IT
```

Figur 4 Menyn med val

```
Select what faults you want to ammend:

1. Skip intro sequence = False
2. Text Speed = 3
3. Return
```

Figur 5 Settings med val

#### 4.4 Karaktär och Spelnivå Generation

Detta spel behöver en spelkaraktär samt en spelnivå för att stödja den tänkta funktionaliteten. För detta så skapas en klass som heter karaktär med olika variabler såsom "hp" och "strength" som kan användas och refereras till utav alla karaktärsrelaterade funktioner.

Men utöver en karaktär så behövs det dessutom en spelplan. Spelplanen implementeras genom att det genereras en slumpmässig nivå med olika möbler och objekt. Utöver miljö-objekt så ska det alltid finnas en fiende att besegra. Dessa fiender läses in ifrån i en JSON fil som innehåller alla möjliga fiender.

Varje fiende har en svårighetsnivå som korrelerar till den nivå de kan finnas på, till exempel så kan en "Goblin" bara vara på nivå ett. Dessa fiender har egna värden såsom "hp" och "damage" samt en "defense type" som används senare i "combat" delen utav projektet.

Chattbotten ChatGPT användes för att generera basgruppen utav monster som finns i denna projektversion. Anledningen till detta är för att snabba upp arbetsprocessen då en kvantitet utav fiender behövdes, men kvaliteten är relativt o-viktig.

Monstret sparas sedan i ett monster-objekt som sparas i en nivå-objekt som slutligen sparas direkt i karaktär klassen. Denna struktur tillåter hela spelets status att sparas genom att bara spara karaktär klassen.

```
{
  "name": "Goblin",
  "description": "A small, green-skinned creature that loves to ambush travelers.",
  "hp": 4,
  "damage": 1,
  "defense_type": 0,
  "difficulty_tier": 0
},
{
  "name": "Troll",
  "description": "A giant, brutish creature that regenerates health slowly.",
  "hp": 16,
  "damage": 4,
  "defense_type": 1,
  "difficulty_tier": 2
},
}
```

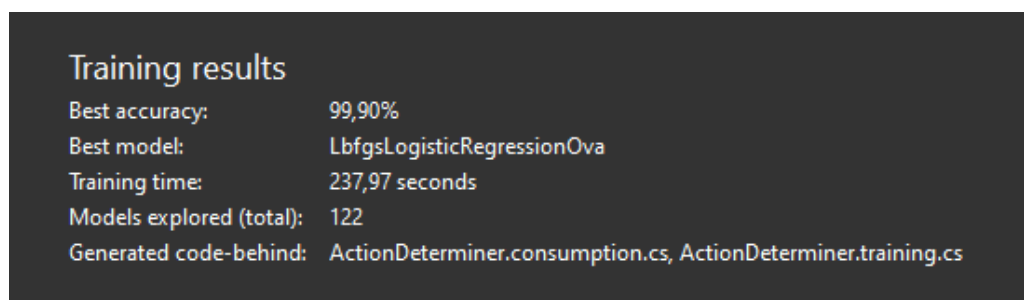
Figur 6 Två monster exempel

## 4.5 MLM

Som detaljerat i problemlösningen så ska MLM användas för att tolka användarinput och sedan välja gärnings utfall utefter detta. För detta syfte så skapas en modell för kategoriseringen utav data. Denna modell kräver en stor mängd rå-data för att träna. Varefter den kan kategorisera stycken med den tränade datan som referens. För att skapa en stor mängd data utan att spendera en stor mängd tid så användes ChatGPT till detta.

För varje möjlig gärning såsom "rest" eller "open chest" så frågades ChatGPT "ge mig 200 exempel utav 'x'" där x är till exempel "sits down and rests". Därefter så filtreras all den informationen och tilldelades ett numeriskt värde baserat på gärningen, varefter det kan användas för träning.

Varje aktion kräver ungefär 200 rader av rå-data för att modellen ska kunna igenkänna det väl. Denna data matas sedan in i MLM för att träna den. Det tog ett par försök och omformeringar utav träningsdatan för att få modellen att bli någorlunda pricksäker i sina svar.



Figur 7 Träningsresultat

Den slutliga modellen fungerar i nästan alla utfall, och är bra nog för projektets ändamål. För att förbättra den så skulle det behövas stora mängder data som inte är värt tiden det skulle ta att generera. Däremot så blev pricksäkerheten skarpt försämrade när antalet olika gärnings utfall var över 5. Därför splittrades modellen upp i två olika, en för generella gärningar och en som bara används i "combat" delen.

## 4.6 Spel, Sparning och Combat

När alla grundläggande funktioner är implementerade så kan dessa läggas ihop för att skapa själva spelet. Spelet fungerar genom att nivån ifrån karaktärens klass läses in med alla dess objekt och monster. Varefter karaktären får göra en gärning såsom "rest" eller "look around" för att interagera med sin omgivning. Med slutmålet på varje nivå att gå i strid med monstret och besegra den, och ifall den besegras så kan spelaren gå till nästa nivå.

När en nivå avklaras så ska en ny nivå generas, då kallas generatorn med nuvarande nivåns nummer för att generera en nivå med ett nummer över denna.

Objekten på spelnivån har alla någon form av potentiell interaktion med spelaren. Till exempel så kan en kista öppnas och med lite tur så kan till exempel ett nytt vapen hittas. Vapen bestämmer karaktärens "damage" med olika "typer" beroende på vapnet. Till exempel så är ett spjut typen "1" medan svärd har "0". Dessa typer ger spelaren en bonus om dem slåss mot monster utav samma defence type som spelarens vapen.

För att kunna spara spelet och använda ytterligare JSON manipulation i projektet så kopieras karaktär klassen med all dess information till en JSON fil när varje nivå skapas. Och eftersom nivån är sparad i karaktären så blir en fullständig kopia av spelets status lagrad för att kunna fortsätta rakt där användaren avslutade. Alla sparnings funktionaliteter är helt automatiska, vilket skapar mindre att tänka på för användaren.

Denna JSON sparfilms information kan även användas när spelet startas om, till exempel om filen finns med en nivå sparad så behöver inte intron spelas då spelaren redan har sett den.

Spelets "combat" eller strid sker när användaren väljer att gå till monstret och starta detta läge. Detta händer i formen av en while sats som inte slutar förrän monstret eller spelaren har "dött". I "combat" så får spelaren göra en gärning som antingen är en "piercing attack" eller "slashing attack".

Dessa attacker avgör hur mycket skada spelaren gör baserat på tur (slumpmässiga nummer), fiendens "defence type" och spelarens vapen. Ifall monstret inte dör får den attackera tillbaka efter spelaren har attackerat.

När spelarens eller monstrets liv når 0 sluter "combat" läget och ifall spelaren överlevde så kan hen gå till nästa nivå och får lite högre "strength" som avgör gränsen för "hp". På så vis så får spelaren möta farligare motståndare och bli starkare för varje nivå. Sista nivån är nivå 3 varefter spelet vinnas.

```
You are in an square room with a narrow hallway leading
forward... At the end of the passage you see another
room and a Vampire Bat, looking menacingly at you.

look around

You look around the room...

A rudimentary wooden table dominates the room. The
table is relatively featureless and has a quaint Wooden
Chair that accompanys it.
```

Figur 8 Start nivå där "look around" har skrivits in för att köra "look around" gärningen.

## 5 Resultat

Det resulterande spelet uppfyller alla projektets mål till dess fullständighet. Spelet går att spela, det har ett unikt grafisk stil, med text som försöker skapa en bra "atmosfär" och skrivs ut stilistiskt. Spelet innehåller JSON funktionaliteten både i formen av att läsa monster data ifrån en separat fil och kring sparningen och användningen utav karaktärs-filen.

Spelets sparfil uppdaterar även dess data genom att skriva över sparningen på varje nivå och uppfyller därför alla utsatta mål med JSON manipulation.

Spelet har strukturerats med en tydlig struktur där användandet av en global karaktär-klass och settings-klass enkelt låter funktioner och klasser använda dess data. Programmet kan därför anses som strukturerat och sammanhängande.

Slutligen så har det även lyckats att använda MLM för att kunna analysera användarinmatning med en någorlunda hög grad av säkerhet. Majoriteten av spelets interaktioner händer genom inmatningar ifrån användaren som tolkas av en MLM. På grund av denna breda och lyckade användning så anses MLM målet som uppfyllt.

Spelprototypen uppfyller alla mål och löser alla utsatta problem som detaljerades inom introduktionen. Därav anses projektet i sin helhet som lyckat.



## 6 Slutsatser

Personligen så är jag väldigt nöjd med projektets resultat, den har däremot en del saker som jag inte är helt nöjd med. Men funktioner/saker är tyvärr inte värt tiden eller arbetet att lösa. Alla grundfunktionerna i spelet fungerar och det behövs inte så mycket jobb för att kunna anse att det är ett underhållande "litet" spel. Främst bara mer volym av innehåll, men detta visar inte på någon teknisk förmåga och är därför slöseri med tid för denna uppgift.

Är väldigt nöjd med integrationen och funktionaliteten utav MLM i projektet, blev förvånad hur bra den blev men samtidigt så önskade jag att den vore mer pricksäker. Detta skulle kunna fixas om jag hade någon bra källa för en "stor" volym av träningsdata. Men jag anser att den är tillräckligt pricksäker för projektets syften.

Är dock inte särskilt nöjd med hur jag hanterar spelets "text", allt är "råskrivet" inom block rakt i koden. Vilket är en massiv inflation utav kodstorleken. Men att formatera om allt till att läsa filer och inse vart radbrytningar och färgläggningar skulle vara är en hel del arbete som jag inte vill lägga på en redan sen uppgift. Men ifrån användarperspektivet så spelar det i alla fall ingen roll om hur det görs. (såvida inte prestanda tar stryk)

Rent grafiskt så ville jag imitera min "vision" utav gamla text RPG:s innan spelgrafik och miljöer blev brett tillgängligt. Nu har jag aldrig spelat ett sådant spel själv, men jag ansåg att detta var en bra möjlighet att använda MLM och JSON till att göra något faktiskt unikt och kreativt.

Tidigare inom olika liknande projekt har det alltid varit, "skapa någon form av interaktiv lista". Men jag ansåg att detta inte var stimulerande nog så gick mig in på något som krävde lite mer tänk. Tack vare detta så fick jag eftersöka och testa en mängd olika funktionaliteter som jag normalt inte rör.

Hur gör jag göra en bra "paus"? Ska man kunna hoppa intron? När bör konsolen rensas? Hur gör jag det mer interaktivt? Är bara några exempel på de frågor jag behövde besvara och lösa under projektets gång.

Strukturellt var det också utmanande, hur ska jag spara spelardata utan att kasta runt 5 dupliceringar av samma objekt som kräver uppdatering vid varje ändring? Min lösning till detta var statiska publika klasser, vilket jag inte är helt nöjd med men det fungerar. Gör det lite jobbigare att spara dock då statiska variabler ifrån klasser inte skapas när man skapar ett nytt objekt av samma klass.

Stridssystemet är också någorlunda intressant då jag försökte få det så "roligt" som möjligt utan att skapa något simpelt "större nummer vinner" system. Men det kan förbättras mycket och skulle jag ha lagt mer tid på projektet så skulle fokusen varit på att göra "combaten" till mer

Konsol Spel – Enkelt Text Baserat RPG

Max Gagner (maga2101)

2024-12-01

än "slash/thrust". Men detta skulle behöva båda nya funktioner och framför allt hårda ändringar i combat MLM'en och dess träningsdata.

## Källförteckning

1. Microsoft "what is a Machine Learning Model"  
<https://learn.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model> 2024-11-28
2. Microsoft "using Threads and Threading"  
<https://learn.microsoft.com/en-us/dotnet/standard/threading/using-threads-and-threading> 2024-11-28
3. Microsoft "Visual Studio" <https://visualstudio.microsoft.com/> 2024-11-28
4. Git "Git" <https://git-scm.com/> 2024-11-28
5. OpenAI "ChatGpt" 4o-mini modellen har använts till uppgiften  
<https://chatgpt.com/> 2024-11-29
6. ASCII Art "Text to ASCII art" <https://www.asciiart.eu/text-to-ascii-art> 2024-11-29