| Name: | | | |
| --- | --- | --- | --- |
| Henrik Kjernmoen Gran | | | |
| Date: | Course Code: | Course Name: | # of pages: |
| 05/01/2025 | IDATT2505 | Extended Reality (XR) | TODO |

|  |
| --- |
| Lecturers: |
| Thomas Holt, Alexander Holt |

|  |
| --- |
| Title: |
| Report for portfolio, IDATA2505 Extended Reality (XR) |

|  |
| --- |
| Summary: |
| Denne rapporten handler om løsningen min på mappevurderingsoppgaven som ble gitt i Programmering 2. Oppgaven går ut på å lage en historiebasert spillmotor, noe som jeg har løst ved å lage et dataspill hvor man kan lage egne historier, og spille gjennom uendelig mange historier laget av andre bruke. Rapporten beskriver fremgangsmåten mitt design og min implementasjon av dette spillet, og konkluderer med litt drøfting og en liten konklusjon om spillet jeg har laget. |

*This report is a submission by student(s) at NTNU.*

INNHOLD

[1 SAMMENDRAG 1](#_Toc135722727)

[2 Begreper og forkortelser 1](#_Toc135722728)

[3 Introduksjon 1](#_Toc135722729)

[3.1 Bakgrunn 1](#_Toc135722730)

[3.2 Avgrensninger 1](#_Toc135722731)

[3.3 Begreper/Ordliste 1](#_Toc135722732)

[4 Teori 2](#_Toc135722733)

[4.1 Kobling (Coupling) 2](#_Toc135722734)

[4.2 Kohesjon (Cohesion) 2](#_Toc135722735)

[4.3 Design-mønster (Design patterns) 2](#_Toc135722736)

[4.3.1 Observer pattern 2](#_Toc135722737)

[4.3.2 Builder pattern 3](#_Toc135722738)

[4.3.3 Model-view-controller pattern (MVC) 3](#_Toc135722739)

[4.4 Enhetstesting 3](#_Toc135722740)

[5 Kravspesifikasjon 4](#_Toc135722741)

[5.1 Funksjonelle krav 4](#_Toc135722742)

[5.1.1 Link-klassen 4](#_Toc135722743)

[5.1.2 Passage-klassen 5](#_Toc135722744)

[5.1.3 Story-klassen 6](#_Toc135722745)

[5.1.4 Player-klassen 7](#_Toc135722746)

[5.1.5 Game-klassen 7](#_Toc135722747)

[5.1.6 Action-interfacet og Action-klasser 8](#_Toc135722748)

[5.1.7 Goal-interfacet og Goal-klasser 8](#_Toc135722749)

[5.2 Krav til filhåndtering 9](#_Toc135722750)

[5.3 Krav til brukergrensesnitt (GUI) 9](#_Toc135722751)

[5.4 Tekniske krav 9](#_Toc135722752)

[6 Teknisk Design 10](#_Toc135722753)

[7 Utviklingsprosess 10](#_Toc135722754)

[7.1 Idémyldring 10](#_Toc135722755)

[7.2 Programmering av modellen 10](#_Toc135722756)

[7.3 Programmering av GUI 10](#_Toc135722757)

[8 Implementasjon 11](#_Toc135722758)

[8.1 Implementasjon av modellen 11](#_Toc135722759)

[8.2 Implementasjon av GUI 12](#_Toc135722760)

[9 Testing 13](#_Toc135722761)

[10 Utrulling til sluttbruker (deployment) 13](#_Toc135722762)

[11 Drøfting 13](#_Toc135722763)

[12 KONKLUSJON - ERFARING 14](#_Toc135722764)

[13 REFERANSER 14](#_Toc135722765)

[14 Vedlegg 15](#_Toc135722766)

[14.1 Brukerveiledning 15](#_Toc135722767)

[14.2 gameFileStructure.md 15](#_Toc135722768)

Figur-liste

[Figur 1 Model-view-controller pattern 3](#_Toc135710556)

[Figur 2 Paths-spill – Eksempel på spillebrett 4](#_Toc135710557)

[Figur 3 Krav til Link-klassen 4](#_Toc135710558)

[Figur 4 Krav til Passage-klassen 5](#_Toc135710559)

[Figur 5 Krav til Story-klassen 6](#_Toc135710560)

[Figur 6 Krav til ekstra metoder i Story-klassen 6](#_Toc135710561)

[Figur 7 Krav til Player-klassen 7](#_Toc135710562)

[Figur 8 Krav til Game-klassen 7](#_Toc135710563)

[Figur 9 Krav til Action-interfacet med implementerende Action-klasser 8](#_Toc135710564)

[Figur 10 Krav til Goal-interfacet med implementerende Goal-klasser 8](#_Toc135710565)

[Figur 11 Krav til format på .paths-fil 9](#_Toc135710566)

[Figur 12 Klassediagram av modellen (modelClassDiagram.drawio) 11](#_Toc135710567)

[Figur 13 Aktivitetsdiagram av applikasjonen (activityDiagram.drawio) 12](#_Toc135710568)

Tabell-liste

[Tabell 1 Begreper og ordliste 2](#_Toc135710608)

# Summary

Denne rapporten handler om løsningen min på mappevurderingsoppgaven som ble gitt i Programmering 2. Oppgaven går ut på å lage en historiebasert spillmotor, noe som jeg har løst ved å lage et dataspill hvor man kan lage egne historier, og spille gjennom uendelig mange historier laget av andre bruke. Rapporten beskriver fremgangsmåten mitt design og min implementasjon av dette spillet, og konkluderer med litt drøfting og en liten konklusjon om spillet jeg har laget

# Begreper og forkortelser

LTS Long Time Support

MVC Model-View-Controller pattern

MVP Minimum Viable Product

# Introduction

## Background

This report is regarding the portfolio submitted for evaluation in the Extended Reality (XR) course. The main purpose of this project is to cover the curriculum of the Extended Reality (XR) course, but since the actual requirements for this project were very open, it was also combined with an earlier passion project of mine, that being an infinite Minesweeper game. The earlier project was made without a proper game engine & with very little structure, so this new project became a sort of re-make, as well as an expansion of the original project into a roguelike, so it would be a “new” project & not just a pure remake of an existing one.

## Restrictions

* One of the main restrictions of this project is time. Due to being quite an ambitious project, there’s a lot of potential features that could be added to the game. Therefore, the limiting factor for many ideas is if it’s worth the effort of implementing, or if that time is better spent working on something else.
* Certain relevant topics were a little outside the scope of this course’s curriculum, like multithreading, sound design, UX design, Game design/theory, etc. These topics have therefore not been prioritized as much compared to their normal level of priority in game development.
* Due to the game being based on Minesweeper, a 3D implementation wasn’t very feasible, as Minesweeper is not a game that translates well into 3D. While it would’ve been possible to make a 3D version of Minesweeper in VR or something like it, that would’ve been more a “proof of concept” rather than an actual enjoyable game. Therefore, the decision was made (with approval) to stick to 2D, for the sake of the game not being needlessly complicated & clunky.

## Begreper/Ordliste

| Begrep (Norsk) | Begrep (Engelsk) | Betyding/beskrivelse |
| --- | --- | --- |
| Skjerm | Screen | En unik scene med elementer som brukeren kan interagere med, og som kan vises inni applikasjonsvinduet. Består av «view»- og «controller»-delen av MVC. |
| Modell | Model | Selve spill-logikken som styrer spillet. «model»-delen av MVC |
| Spill | Game | Et spill som kan lastes inn av brukeren for å spilles |
| Spiller | Player | Selve brukeren som spiller spillet, representert i applikasjonen |
| Historie | Story | Selve historien som tilhører et spill |
| Fiende | Enemy | Diverse fiender som tilhører et spill, og som spilleren kan kjempe mot |
| Inventar | Inventory | En liste med gjenstander som spilleren og fiender kan holde på |
| Gjenstand | Item | Et objekt som lagres i et inventar, og som spilleren og fiender kan interagere med |

Tabell 1 Begreper og ordliste

# Theory, technology & architecture

## Game engine

The first question that needed to be answered for this project was what game engine to use. With many great options to choose from, Godot was chosen, due to being entirely free & open source, easy to set up, and simple & intuitive to use. The one drawback with Godot is that it does not follow the Model-View-Controller design pattern, instead splitting the game into a hierarchy of nodes, each with their own model, view & controller responsibilities. Although despite this unorthodox design, it does have its strengths as well, with a little bit of getting used to it.

## Separate model

A separate model was created without dependence on Godot, leaving only the view & controller responsibilities to Godot in this project. The model was created with a single singleton entry point, that any part of the front-end (in this case, any Godot node script) can access. When changes in the model occur, these changes are then broadcasted to receivers in the front-end through an event system, allowing the UI to update accordingly

## Graphic design

To easily make textures for the game, Aseprite was used. Aseprite is a paid external software that can be purchase on Steam, made specifically for creating pixel art. This software is in no way necessary to evaluate the project, as it just fills the purpose of an image editor & that’s it. Textures are stored in universally recognized image formats, and do not require Aseprite to be viewed or modified.

## Design-mønster (Design patterns)

### Observer pattern

Hvis et objekt kontinuerlig spør et annet objekt om noe helt til det objektet svarer med ønsket resultat, så ender det med at mye tid og ressurser ikke blir brukt til noe nyttig. En god løsning på et slikt problem er «observer pattern», som går ut på at et objekt kan abonnere på et annet objekt, slik at det andre objektet heller kan si ifra når det har abonnentens ønskede resultat. I tillegg trenger objektet som blir abonnert på kun å vite hvordan det skal si ifra til abonnentene sine, noe som gjør det enkelt å holde en løs kobling mellom abonnentene og objektet som blir abonnert på.

### Builder pattern

I tilfeller hvor en klasse får en veldig stor konstruktør med mange parametere, med verdier som ikke alltid trenger å spesifiseres, så bør «builder pattern» benyttes. Builder pattern går ut på av man gjemmer konstruktøren til en klasse, og heller lager en indre builder-klasse. Denne klassen har da en rekke metoder for å sette parametere, som også returnerer selve builder-instansen, slik at man kan koble flere metodekall til builder-objektet sammen. Til slutt har builder-klassen en «build()»-metode, som bygger selve objektet av typen til den ytre klassen, og returnerer det. Dette gjør at man kan lage objekter på en mye ryddigere måte, uten lange og uleselige konstruktører.

### Model-view-controller pattern (MVC)

Når et brukergrensesnitt for en applikasjon skal utvikles, kan dette deles in i 3 deler: Model, view og controller. «Model» er selve kjernen som inneholder logikken til applikasjonen, «view» er det visuelle som brukeren ser, og «controller» er det funksjonelle som brukeren kan interagere med. «Model-view-controller pattern» beskriver hvordan disse delene interagerer med hverandre (se figur 1).

Diagram

Description automatically generated

Figur 1 Model-view-controller pattern

# Kravspesifikasjon

## Funksjonelle krav

Diagram

Description automatically generated

Figur 2 Paths-spill – Eksempel på spillebrett

### Link-klassen

A picture containing text, screenshot, font, number

Description automatically generated

Figur 3 Krav til Link-klassen

### Passage-klassen

A screen shot of a computer program

Description automatically generated with low confidence

Figur 4 Krav til Passage-klassen

### Story-klassen

A picture containing text, screenshot, font, number

Description automatically generated

Figur 5 Krav til Story-klassen

I tillegg skal følgende metoder implementeres ved hjelp av funksjonell programmering og streams:

A screenshot of a computer program

Description automatically generated with low confidence

Figur 6 Krav til ekstra metoder i Story-klassen

### Player-klassen

A screenshot of a computer program

Description automatically generated with medium confidence

Figur 7 Krav til Player-klassen

Det skal også implementeres builder-pattern for Player-klassen

### Game-klassen

A screen shot of a computer program

Description automatically generated with low confidence

Figur 8 Krav til Game-klassen

### Action-interfacet og Action-klasser

A picture containing text, screenshot, diagram, font

Description automatically generated

Figur 9 Krav til Action-interfacet med implementerende Action-klasser

### Goal-interfacet og Goal-klasser

A screenshot of a computer

Description automatically generated with medium confidence

Figur 10 Krav til Goal-interfacet med implementerende Goal-klasser

## Krav til filhåndtering

Programmet skal ha følgende støtte for filhåndtering:

* Det skal være mulig å lagre en historie i en fil
* Det skal være mulig å lese en historie fra en fil
* Fila skal være en tekstfil og følge et bestemt format (se Figur 11)
  + En ny passasje starter med :: (to semikolon) og passasjens tittel
  + Den neste linjen holder passasjens innhold (content)
  + De siste linjene i en blokk lister opp linkene
  + Hver link får sin egen linje og er på formen [text](reference)
  + En blokk avsluttes med en tom linje
* Filendelsen skal være «.paths»

A picture containing text, screenshot, software, font

Description automatically generated

Figur 11 Krav til format på .paths-fil

## Krav til brukergrensesnitt (GUI)

* Det skal være mulig å laste inn en historie fra en .paths-fil.
* Brukergrensesnittet bør opplyse om filens navn, lokasjon og evt døde linker.
* Før man starter et spill må man kunne legge inn informasjon om en spiller (navn, helse og gullbeholdning).
* Før man starter et spill må man kunne legge inn goals. Man skal kunne legge til så mange goals man vil.
* Det skal være mulig å gjennomføre et spill fra start til slutt.
* Spillerens helse, poengsum, gullbeholdning og inventar skal være synlig gjennom hele spillet.
* Det må være mulig å restarte et spill. En restart tar spilleren tilbake til første passasje og nullstiller helse, poengsum, gullbeholdning og inventar.
* Brukeren må kunne få informasjon om hvordan man spiller (brukerveiledning/hjelp).
* Brukergrensesnittet skal ha et oversiktlig layout med tydelige komponenter, god kontrast og fornuftige feilmeldinger.

## Tekniske krav

* Prosjektet skal kunne bygges med «mvn clean» og «mvn package» uten feil
* Prosjektet skal kunne kjøres med «mvn javafx:run»

# Teknisk Design

Det ble valgt at det skulle lages en desktop-applikasjon, ettersom kravspesifikasjonen krevde en applikasjon med et grafisk brukergrensesnitt som kan bygges i Maven. Det stilles ingen krav til nettbasert flerspiller, skylagring, databaser eller annen funksjonalitet som krever noe mer enn kun en desktop-applikasjon, og dermed ble det valgt som systemarkitekturen.

Applikasjonen er satt opp og programmert som et Maven-prosjekt, ettersom kravspesifikasjonen krevde dette. I tillegg er Java 17 benyttet, noe som også var et krav, ettersom det er siste LTS-versjonen av Java.

# Utviklingsprosess

## Idémyldring

Det aller første som ble gjort var en skikkelig idémyldring til hva som kunne bli lagt til av funksjonalitet, og hva slags spill-sjanger den endelige applikasjonen skulle være. Flere alternativer til spill-sjanger ble vurdert, blant annet historiedrevet spill, rogue-like, puzzle-game, og så videre. Til slutt ble det en historiesimulator som ble valgt, hvor applikasjonen har masse funksjonalitet uten innhold, hvor man selv kan lage innholdet og deretter spille gjennom det innholdet man har laget som en historie. Dette gjorde at applikasjonen ble en slags spillmotor for å spille gjennom historier laget av andre spillere. Med dette ble videre utvikling kun fokusert på funksjonalitet, uten noe særlig fokus på innhold.

Det ble vurdert mange idéer til ekstra funksjonalitet, og blant disse ble det valgt at kampsystem med fiender, interaktive gjenstander, vanskelighetsgrader og mulighet til ekstern lagring av dette skulle bli implementert. Prosjektet startet originalt med noen flere idéer til ekstra funksjonalitet, men disse ble etter hvert flyttet til en egen dev-branch for å eventuelt bli implementert senere, noe som ikke endte opp med å skje. Blant disse idéene var automatisk historiegenerering, tilfeldig generering av gjenstander, butikksystem for å kjøpe eller selge gjenstander, og så videre.

## Programmering av modellen

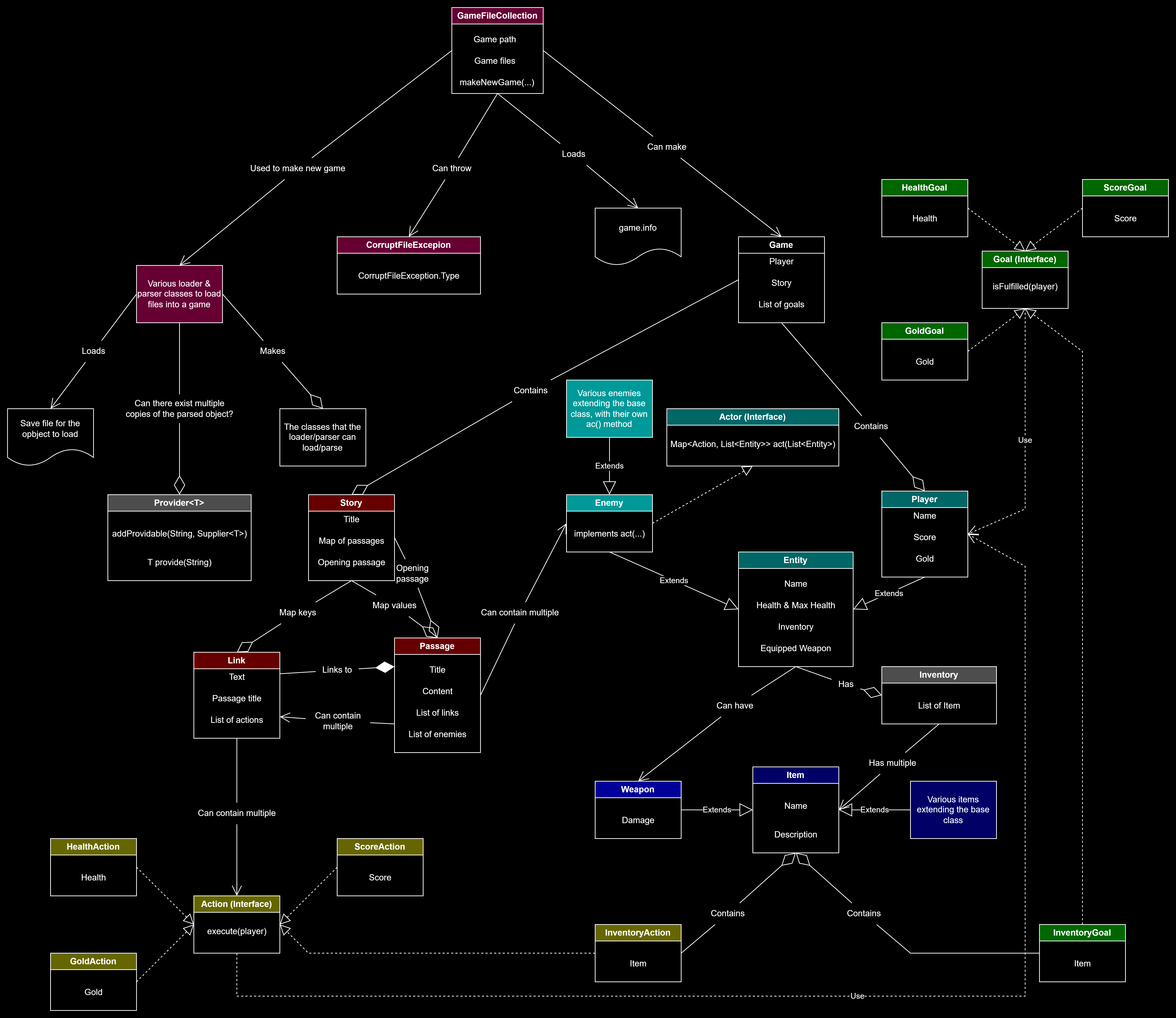
Etter at en generell visjon av den endelige applikasjonen var på plass, ble laget et klassediagram over selve modellen (se Figur 12), og deretter startet utviklingen av selve modellen. Denne utviklingen startet med å få på plass de grunnleggende klassene spesifisert i den funksjonelle delen av kravspesifikasjonen. Etter at det var programmert, begynte utviklingen av ekstra funksjonalitet. En del av denne funksjonaliteten ble etter hvert flyttet over til dev-branchen for senere utvikling, men noe ble beholdt på main-branchen, og ble utviklet videre med resten av applikasjonen. Kravet til filhåndtering ble etter hvert implementert, og enhetstesting ble utført parallelt med resten av utviklingen. Det ble jevnlig gitt tilbakemeldinger av faglærer & læringsassistenter i denne perioden.

## Programmering av GUI

Når selve spill-logikken var på plass, var det neste steget programmering av GUI-en. Dette startet veldig enkelt, men etter hvert ble MVC implementert, og «view»-delen ble utviklet parallelt med «controller»-delen. Når selve spill-skjermen skulle lagres, ble også filhåndteringa utvidet mot det endelige målet om et spill som kan tilpasses med all innebygd funksjonalitet. Det ble også utviklet et mellomledd mellom modellen og resten av applikasjonen, som tok input fra controller, oppdaterte modellen, og sendte tilbake et event ved bruk av observer-pattern. Det ble etter hvert opprettet et aktivitetsdiagram over brukergrensesnittet. Når alt dette var på plass, ble applikasjonen etter hvert ferdigstilt etter en siste runde med bugfixing og opprydding.

# Implementasjon

## Implementasjon av modellen



Figur 12 Klassediagram av modellen  
(modelClassDiagram.drawio)

Selve modellen ble først implementert i henhold til kravspesifikasjonen, men det ble etter hvert noen ekstra tillegg og endringer:

* Item: Representerer en gjenstand. InventoryAction, InventoryGoal & Player.inventory holder på Item istedenfor String.
* Inventory: Representerer et inventar med gjenstander. Player.inventory er av typen Inventory istedenfor List.
* Enemy: En «spiller» kontrollert av datamaskinen som angriper spilleren. En passasje kan ha flere enemies
* Entity: Ettersom spilleren og fiender har mange felter og mye funksjonalitet til felles, blir Entity opprettet som en abstrakt superklasse.
* Actor: Et interface som representerer noe som kan handle i løpet av en kamprunde uten input fra spilleren. Blir implementert av Enemy.
* Flere loadere: Fillagring blir delt opp i flere filer pga. mye data og spesifikk rekkefølge, og Loader-klasser blir opprettet for hver type fil som kan lagres.
* Provider: Kan legge til providable objekter, og kan dele ut kopier av disse. Blir brukt for innlasting av lagrede items og enemies.
* GameFileCollection: En samling av spillfiler, som kan lastes inn i riktig rekkefølge for å lage et nytt spill.

## Implementasjon av GUI

A picture containing screenshot, text, line, design

Description automatically generated

Figur 13 Aktivitetsdiagram av applikasjonen  
(activityDiagram.drawio)

Brukergrensesnittet ble implementert ved hjelp av en GenericScreen-klasse, som er en superklasse for alle skjermene. Dette er en abstrakt subklasse av JavaFX sin Scene-klasse, og den inneholder følgende:

* initializeNoder(): En funksjon som gjør ingenting, men som blir kalt før makeRoot(), og kan implementeres av andre subklasser.
* makeRoot(): En abstrakt funksjon som lager root-noden til scenen.
* setDefaultState(): En funksjon som blir kalt rett før skjermen blir vist i applikasjonsvinduet.
* updateSize(): Transformerer root-noden til å passe på skjermen. Endrer ikke den definerte størrelsen på root-noden, slik at innhold inni den blir beholdt på samme relative plass.

I tillegg er alle kontrollere subklasser av en GenericController-klasse, som har en changeScreen()-metode for å endre den nåværende skjermen til en annen.

# Testing

Testing er hovedsakelig gjort ved hjelp av enhetstesting for modellen til applikasjonen, mens resten er manuelt testet ved å sjekke om det «gir mening». Det testes ikke om applikasjonen kan laste inn filer, men det testes om tolkning av simulerte filer fungerer.

Applikasjonen har ikke blitt brukertestet av brukere som ikke har vært involvert i utviklingen. Dette er hovedsakelig på grunn av tidspress mot slutten, ettersom det ikke ble særlig mye tid igjen til brukertesting.

# Utrulling til sluttbruker (deployment)

Applikasjonen gjøres tilgjengelig for sluttbruker ved at det lages en desktop-applikasjon, og den kan bygges med «mvn clean package», og kan bygges med «mvn javafx:run». Applikasjonen oppfyller alle spesifiserte krav som er stilt til bygging & kjøring.

Applikasjonen krever at brukeren må velge en historie å spille, noe som ikke er inkludert i applikasjonen. Brukeren må derfor lage en historie selv for å spille, eller skaffe en historie fra andre eksterne kilder. Applikasjonen er laget slik ettersom det enkelt lar brukere lage & dele egne historier med hverandre. Historier må deles direkte mellom brukere, ettersom det ikke er satt opp servere for opp- og nedlastning av historier. Dette er hovedsakelig på grunn av at det går utenfor kravspesifikasjonen til applikasjonen.

# Drøfting

Oppgaven ble til slutt fullført, men med ganske mye tidspress mot slutten. Dette førte til at diverse planlagte funksjoner ikke ble implementert, som for eksempel mulighet til å bruke gull til å kjøpe og selge gjenstander, mulighet for kritiske treff ved angrep, og så videre. Problemet med tidspress mot slutten ble derimot lagt merke til tidlig nok til å forhindre problematiske mangler, og den endelige applikasjonen ble ganske så ferdigstilt, med robust kode, god kodestil & gode designprinsipper. Selv om alle idéene til applikasjonen ikke ble implementert, endte de funksjonene som faktisk ble implementert opp med å bli implementert på en god måte med god kvalitet.

Hovedkildene til koden er Objects First With Java[1], undervisningstimer[2] & offisiell Javadoc[3]. Disse kildene er pålitelige, og andre diverse kilder som er brukt til å løse mer spesifikke problemer har blitt sjekket at de stemmer overens med relevante prinsipper beskrevet i hovedkildene. Hovedkildene har stemt ganske godt overens med hverandre, men resten av kildene har noen ganger avviket fra det hovedkildene sier. I disse har kildene blitt utnyttet til en viss grad, men korrigert slik at de stemmer overens med hovedkildene, som i gjengjeld bidrar til god & robust kodekvalitet.

# KONKLUSJON - ERFARING

Utviklingen av applikasjonen startet greit, og den ble utviklet kontinuerlig gjennom hele utviklingsperioden. Effektiviteten på selve arbeidet gikk derimot litt nedover mot slutten av prosjektet, noe som hovedsakelig skyldes manglende arbeidslyst, som igjen skyldes min mors død mot slutten av prosjektet. Min mor døde relativt brått i slutten av april, noe som i stor grad påvirket motivasjonen til å jobbe på mappeoppgaven, og gjorde at en del ønsket funksjonalitet ikke ble lagt til på grunn av dårlig tid. Om jeg kunne ha begynt oppgaven på nytt uten at dette hadde skjedd, tror jeg at jeg kunne ha fått til mer enn jeg endte opp med å få til nå.

Når det er sagt, så realiserer den endelige applikasjonen hele kravspesifikasjonen til MVP, og det som ble gjort fikk godt kvalitet til slutt. I noen tilfeller tok det derimot noen runder med refaktorering før kvaliteten kom på plass, som igjen påvirket kvantiteten med funksjonalitet som ble lagt til. Kvalitet ble derimot prioritert over kvantitet, slik at spilleren ikke møter på uventede begrensninger på grunn av dårlig spagetti-kode.

Spillet har derimot noen uvanlige begrensninger, men disse kommer fra bugs i JavaFX-biblioteket, og ikke koden som har blitt skrevet som en del av mappevurderingen. Disse er som følger:

* ScrollPane oppdaterer ikke størrelsen på riktig måte når scrollbars vises, og scrollbar policy er satt som «as-needed»[8]
* Endring av scene i fullscreen skrur av fullscreen[9]. Dette er løst ved å gjøre vinduet fullscreen igjen med en gang etterpå, men det skaper merkbar «screen flickering»
* Endring av scenestørrelse tar ikke hensyn til Windows UI zoom rett etter at scenen har blitt satt som scenen til hovedskjermen

For fremtidig arbeid så er ekstra funksjonalitet noe som definitivt bør bli videreutviklet. Applikasjonen sliter litt med at spill fort blir kjedelige, og den ville nok neppe gjort det særlig bra på spillmarkedet i dag. Dette var for så vidt ikke et realistisk mål for denne mappeoppgaven, men hvor «gøy» spillet faktisk er bør definitivt tas i betraktning hvis det skal videreutvikles. Ellers så vil en game editor med GUI gjøre det enklere for spillere å lage egne spill.

# REFERANSER

1. ”Objects First With Java”, Sixth edition, av Barnes og Kölling. ISBN 978-1-292-15904-1
2. Undervisningstimer m/Arne Styve, Dataingeniør Ålesund, vår 2023
3. <https://docs.oracle.com/javase/8/docs/api/>
4. <https://www.stackoverflow.com/>
5. <https://www.w3schools.com/>
6. <https://www.baeldung.com/>
7. <https://www.geeksforgeeks.org/>
8. (JDK-8199934) ScrollPane: horizontal scrollBar hides content, <https://bugs.openjdk.org/browse/JDK-8199934>
9. (JDK-8089209) Replacing scene in stage switches off fullscreen mode, <https://bugs.openjdk.org/browse/JDK-8089209>

# Vedlegg

## Brukerveiledning

For å starte et spill:

1. Play Game -> New Game
2. Skriv inn navn
3. Velg et spill. Et spill er definert som en mappe som inneholder «game.info». Referer til «gameFileStructure.md» for hvordan et spill er bygd opp, og hvordan man lager ett.
4. Next
5. Velg vanskelighetsgrad
6. Play

I spillet:

* Angrip & drep fiender for gull, poeng og kule gjenstander
* Beveg deg i rundt for å oppdage nye områder
  + Fiender kan hindre deg fra å bevege deg, vær forsiktig
* Oppnå alle målene for å vinne

## gameFileStructure.md

Denne fila er i Markdown-format. Den ligger også i Github-repositoriet

<h1>Game directory</h1>  
<h3>Must contain:</h3>  
  
- gamedirectory/game.info  
- gamedirectory/\*.difficulties  
- gamedirectory/\*.goals  
- gamedirectory/\*.paths  
- gamedirectory/\*.player  
  
<h3>Can contain:</h3>  
  
- gamedirectory/\*.enemies  
- gamedirectory/\*.items  
- gamedirectory/\*.sprites  
- Image files  
  
<h1>game.info</h1>  
<p>Contains file paths of every file that's part of the game.<br/>  
File paths are relative from the game directory</p>  
<h3>Must contain:</h3>  
  
- (...).difficulties  
- (...).goals  
- (...).paths  
- (...).player  
  
<h3>Can contain:</h3>  
  
- (...).enemies  
- (...).items  
- (...).sprites  
  
<h1>actions</h1>  
<p>Actions don't have their own file, but are defined within various other files for various reasons<br/>  
Items can be referenced here by their name<br/>  
There are 4 types of actions, each can be defined as follows:</p>  
- {Gold: [int]}  
- {Health: [int]}  
- {Inventory: [Item]}  
- {Score: [int]}  
  
  
<h1>difficulties</h1>  
<p>Contains a list of every difficulty in the game.<br/>  
Difficulties are separated by each line, and can be any text string.<br/>  
Blank lines are ignored.<br/>  
Must have at least 1 difficulty.</p>  
  
<h1>enemies</h1>  
<p>Contains a list of every enemy in the game.<br/>  
Items can be referenced here by their name<br/>  
Enemies are separated by a blank line, and are defined as follows:</p>  
  
- !Enemy Name (Required)  
- -**["Basic" | "Vampire"]** (Required, without quotes)  
- Health: **[int]** (Required)  
- Score: **[int]**- Gold: **[int]**- Inventory: **[Item, Item, ...]**- Weapon: **[Item]** (Must be a weapon) (Will be added to inventory upon enemy creation)  
- Drop Chance: **[double]**- Drop Weapon: **[boolean]**- Escape Chance: **[double]**<h1>goals</h1>  
<p>Contains a list of goals for every difficulty in the game.<br/>  
If a difficulty has no goals, it will not be playable.<br/>  
Items & difficulties can be referenced here by their name<br/>  
Goals are defined for each difficulty, with each difficulty being separated by a blank line.<br/>  
There are 4 types of goals, each can be defined as follows:</p>  
  
- #Difficulty  
- Gold: **[int]**- Health: **[int]**- Inventory: **[Item]**- Score: **[int]**<h1>items</h1>  
<p>Contains a list of every item in the game.<br/>  
Items are separated by a blank line, and are defined as follows:</p>  
  
- #Item Name (Required)  
- -**["Basic" | "Weapon" | "Usable"]** (Required, without quotes)  
- Description: **[int]** (Required)  
- Damage: **[int]** (Required) (Only for "Weapon")  
- On Use: **[Action]** (Required) (Only for "Usable")  
  
<h1>paths</h1>  
<p>Contains the story of the game<br/>  
Items & enemies can be referenced here by their name.<br/>  
The story is defined as a list of passages, each separated by a blank line<br/>  
The first passage will be the opening passage, and at least 1 passage is required.<br/>  
The complete file structure is as follows:</p>  
  
- Story title (Required)  
-  
- ::Opening passage name (Required)  
- Passage content (Required)  
- more passage content  
- **[Link Text]** (Link reference) {Action}  
- **[Another link Text]** (Another link reference) {Another action}  
- !Enemy name  
- !Another enemy name  
-  
- ::Another passage name  
- Passage content  
- more passage content  
- **[Link Text]** (Link reference) {Action}  
- **[Another link Text]** (Another link reference) {Another action}  
- !Enemy name  
- !Another enemy name  
-   
- ...  
  
<h1>goals</h1>  
<p>Contains a player for every difficulty in the game.<br/>  
If a difficulty has no player, it will not be playable.<br/>  
Items & difficulties can be referenced here by their name<br/>  
A player is defined for each difficulty, with each difficulty being separated by a blank line.<br/>  
The player can be defined as follows:</p>  
  
- #Difficulty  
- Health: **[int]** (Required)  
- Score: **[int]**- Gold: **[int]**- Inventory: **[Item, Item, ...]**- Weapon: **[Item]** (Must be a weapon) (Will be added to inventory upon player creation)  
  
<h1>sprites</h1>  
<p>Contains a the file path for the sprite of every object in the game that has one<br/>  
Sprites are separated by each line. Blank lines are ignored.<br/>  
Any object can be referenced here by their name.<br/>  
This only does something for enemies & passages.<br/>  
Paths are relative to the game directory.<br/>  
The sprites can be defined as follows</p>  
  
- **[Object name]** : **[Sprite path]**