STUDENTRAPPORT

NTNU

KANDIDATNUMMER(E)/NAVN:
John Ivar Eriksen, Emil Slettbakk

DATO:	FAGKODE:	STUDIUM:	ANT SIDER/BILAG:
23/05/23	IDATT2001	Dataingeniør, BIDATA	11 /

FAGLÆRER(E):
Majid Rouhani, Atle Olsø

TITTEL:
Paths – En spillmotor

SAMMENDRAG:

Denne rapporten tar for seg mappevurderingsprosjektet gitt til studentene i faget IDATT2001 Programmering 2, ved Dataingeniør på Norges teknisk- og naturvitenskapelige universitet. Fagets mål er å lære studentene funksjonell og solid programmering, ved bruk av blant annet kjente programmeringsmønstre. For å vise sine egenskaper, både hva de har lært i semesteret og hva de greier å tilegne seg på egenhånd, ble vurderingsformen «mappevurdering» valgt.

Prosjektet gikk ut på å utvikle en spillmotor til et tekstbasert spill. Det gav studentene mulighet til å prøve ut og eksperimentere med nye og mer avanserte metoder og tilnærminger. Videre var det fokus på å jobbe mer profesjonelt, ved å benytte versjonskontroll via Git, og anerkjente arbeidsmetodikker som Scrum og Kanban.

Resultatet ble en fungerende spillmotor, med forskjellige funksjoner for å presentere en ikke-lineær historie. Til tross for forbedringspotensialet i det innleverte produktet, anser gruppen prosjektet for å ha gitt verdifull erfaring og innsikt i programmering, både med tanke på arbeidsmetodikk og programmeringskunnskap.

INNHOLD

Figurliste

Fant ingen figurlisteoppføringer.

Tabelliste

Fant ingen figurlisteoppføringer.

1 SAMMENDRAG

Denne rapporten tar for seg mappevurderingsprosjektet gitt til studentene i faget IDATT2001 Programmering 2, ved Dataingeniør på Norges teknisk- og naturvitenskapelige universitet. Fagets mål er å lære studentene funksjonell og solid programmering, ved bruk av blant annet kjente programmeringsmønstre. For å vise sine egenskaper, både hva de har lært i semesteret og hva de greier å tilegne seg på egenhånd, ble vurderingsformen «mappevurdering» valgt.

Prosjektet gikk ut på å utvikle en spillmotor til et tekstbasert spill. Det gav studentene mulighet til å prøve ut og eksperimentere med nye og mer avanserte metoder og tilnærminger. Videre var det fokus på å jobbe mer profesjonelt, ved å benytte versjonskontroll via Git, og anerkjente arbeidsmetodikker som Scrum og Kanban.

Resultatet ble en fungerende spillmotor, med forskjellige funksjoner for å presentere en ikke-lineær historie. Til tross for forbedringspotensialet i det innleverte produktet, anser gruppen prosjektet for å ha gitt verdifull erfaring og innsikt i programmering, både med tanke på arbeidsmetodikk og programmeringskunnskap.

2 INTRODUKSJON

2.1 Bakgrunn

Oppgaven ble gitt som prosjekt for mappevurdering til førsteårsstudentene ved Dataingeniør BIDATA, NTNU Trondheim. Problemstillingen var å lage en spillmotor for tekstdrevet, ikke-lineære, spill. Det overordnede målet var at motoren skulle utvikles til en tilstand hvor en historie kunne importeres og spilles gjennom fra start til slutt. Oppgaven ble gitt i tre deler, hvor hver bygget på instruksene gitt i den forrige. De to første delene gav detaljerte oppsett av klassene som skulle brukes for å løse oppgaven. I del 3 ble oppgavens rammer løsnet, slik at studentene sto friere til å avslutte og løse oppgaven etter egen evne, skjønn og oppfinnsomhet.

I oppgavene ble det gitt kravspesifikasjoner til programmet. Disse kan ses i mer detalj i kapittel 5 “Kravspesifikasjoner”.

2.2 Avgrensninger

Prosjektet ble gitt enkelte teknologiske avgrensninger i form av hvilket programmeringsspråk og tilhørende komponenter, og versjonskontrollsystem som skulle benyttes:

- Programmeringsspråk: Java, JDK 17 LTS.
- Byggeverktøy: Maven.
- Det grafiske brukergrensesnittet skulle skrives direkte i JavaFX.

- Versjonskontroll: NTNU GitLab.
- Enhetstesting: JUnit.

2.3 Begreper/Ordliste

Begrep (Norsk)	**Begrep (Engelsk)**	**Betyding/beskrivelse**
	UML	Unified Modeling Language
	OOP	Object Oriented Programming
	UX	User Experience
	UI	User Interface
	GUI	Graphical User Interface
	Wireframe	Tidlig modell av GUI. Brukes brukertest og planlegging. Ofte interaktiv, men ikke nødvendigvis.
	JVM	Java Virtual Machine
	JavaFX	Grafisk rammeverk for Java-applikasjoner. Utviklet av Oracle.
	FXML	Markup-språk for å bygge GUI i JavaFX via «style sheets».
	IDE	Interactive Development Environment
	IntelliJ IDEA	IDE for Java, laget av JetBrains.
	git	Version control system
	Maven	Apache Maven is a software project management and comprehension tool.
	Singleton pattern	Software design pattern that restricts the instantiation of a class to a singular instance.
Refaktorisering	Refactoring	**Refactoring** consists of improving the internal structure of an existing program's source code, while preserving its external behavior

3 TEORI

I objektorientert programmering (OOP), er coupling og cohesion to sentrale konsepter. Cohesion refererer til hvor tett relatert funksjonene innenfor en klasse eller modul er. Klasser med høy cohesion har funksjoner som jobber tett sammen mot et felles formål, mens klasser med lav cohesion har funksjoner som er mer uavhengige og ikke nødvendigvis arbeider mot en felles oppgave.

Coupling, derimot, beskriver graden av avhengighet mellom forskjellige klasser eller moduler. Ved høy coupling er klassene tett knyttet sammen, slik at endringer i en klasse kan påvirke andre. Med lav coupling er klassene mer uavhengige, slik at endringer i en klasse har minimal effekt på de andre (GeeksforGeeks, 2023).

I OOP er målet derfor å oppnå høy cohesion og lav coupling. Dette bidrar til å lage et program som er enklere å vedlikeholde, forstå og videreutvikle.

Et mye brukt design mønster når man skal programmere komplekse objekter er "Builder pattern". Ved bruk av Builder pattern deler man opp konstruksjonen av et objekt i fire mindre konstruksjoner. Disse fire blir kalt Product, Builder, ConcreteBuilder og Director. Product er basen og selve objektet som skal bygges, Builder er en abstrakt klasse som definerer trinnene for å bygge produktet. ConcreteBuilder er selve byggeren som implementerer trinnene fra Builder til å bygge produktet. Til slutt har man Director som er ansvarlig for å bygge objektet og bruker de tidligere konstruksjonene for å oppnå dette.

Fordelene med Builder-designmønsteret inkluderer mer robust kode, siden komplekse objekter bygges ved hjelp av små, enkle objekter, og bedre kontroll over konstruksjonen av komplekse objekter. Det bidrar også til å redusere antall parametere i konstruktørene og unngår behovet for å ha mange konstruktører med forskjellige parametere noe som gjør kodene enklere å lese (GeeksforGeeks, 2022).

Abstract Factory Pattern er et annet mye brukt designmønster i programmering, og spesielt nyttig når du arbeider med store og komplekse programvaresystemer som krever å lage forskjellige grupper med relaterte objekter. I Abstract Factory Pattern har vi en overordnet fabrikk, kalt "Abstract Factory", som tillater opprettelsen av andre fabrikker. Hver av disse fabrikkene kan generere en gruppe relaterte objekter uten å spesifisere deres konkrete klasser. Disse gruppene med relaterte objekter kalles ofte "produktfamilier".

Fordelene med Factory-designmønsteret er hovedsakelig at det gir lav coupling ved å skjule de konkrete klassene fra klienten (GeeksforGeeks, 2022).

Singleton er enda et designmønster i som gjør det mulig å sikre at kun en forekomst av en klasse blir opprettet. En instans av klassen lagres som en privat statisk variabel. Metoden som returnerer instansen kalles ofte 'getInstance'. Dette sikrer at bare en instans kan eksistere, og at alle referanser til klassen går gjennom denne metoden.

Fordelen med Singleton Pattern er at det gir en kontrollert tilgang til den enkelte instansen og forhindrer flere instanser i en flertrådet miljø. Men hvis den ikke blir implementert riktig kan den skape feiler i koden (GeeksforGeeks, 2023).

4 KRAVSPESIFIKASJON

Kravene som ble satt frem i del 1 av oppgaven innebar at studenter skulle jobbe i par for å utvikle en programvare som fungerer som en spillmotor for å fremføre en interaktiv historie. Denne programvaren skulle være utformet slik at spillerens valg kunne påvirke historiens utvikling. I tillegg skulle prosjektet integreres med Maven for byggestyring og Git for versjonskontroll.

Del 2 av oppgaven spesifiserte krav til programmets filhåndtering, samt detaljerte hvilke klasser og innhold som skulle være inkludert i den endelige versjonen. Ytterligere krav ble satt til programvarens grafiske brukergrensesnitt (GUI), med en liste over nødvendige funksjoner og designelementer.

Den tredje og siste delen av oppgaven stilte krav om bruk av Builder designmønsteret for å opprette forskjellige spiller objekter i programvaren. Denne delen sikter til å implementere en effektiv og robust måte å opprette og håndtere spiller objektene på.

5 TEKNISK DESIGN

Programmet ble utviklet som en "offline" applikasjon, som kjører lokalt på datamaskinen. Dette fordi oppgaven var å lage en spillmotor til et tekstbasert spill, og dermed var det mest hensiktsmessig kjøre det lokalt. Ingen ressurser må hentes eksternt, siden alt er pakket i kildekoden, med mindre en bruker ønsker å utvide funksjonaliteten.

6 UTVIKLINGSPROSESS

I utviklingen av dette prosjektet ble Kanban metoden valgt som prosessmodell. Kanban er en smidig metodikk som passer godt til prosjekter som tillater raske endringer i programmet ved nye kravspesifikasjoner eller lignende. Dette prosjektet ble delt inn i 3 deler hvor hver del hadde nye krav til programmet, på grunn av dette ble Kanban en passende prosessmodell.

Hvis man kombinerer Kanban metoden med "issue board"-funksjonen til GitLab får man et klart overblikk over prosjektstatus på et gitt tidspunkt hvor hvert kort representerer en oppgave, eller "issue", som skal løses. Noe som sammen tillater en effektiv håndtering av arbeidsmengden og rask respons på endringer.

For hver iterasjon av programmet ble nye issues laget og hvert team medlem hadde oversikt over hva de selv skulle jobbe med og hva den andre holdt på med. I veiledningsmøtene med studentassistent kunne man kjapt ta frem det man har jobbet med siden sist og få detaljert tilbakemelding basert på issue-beskrivelsen i GitLab og selve koden fra IDE.

7 IMPLEMENTASJON

Programmet ble utviklet ved hjelp av IDE-et JetBrains IntelliJ IDEA, og er skrevet i Java (Java Development Kit versjon 17.0.6 LTS). Det benytter byggeautomasjonsverktøyet Maven versjon 3.6.3, i tillegg til JavaFX versjon 17.0.1 for å håndtere konstruksjon av det grafiske brukergrensesnittet. Enhetstesting gjøres ved hjelp av JUnit versjon 5.8.1. Versjonskontroll via git, og benytter NTNU GitLab.

Det ble tidlig besluttet å benytte Singleton-mønsteret, siden mange av klassene som representerer objekter som bør eksistere i en gitt tilstand gjennom hele programmets kjøretid. Spesielt dreier dette seg om Main Menu, New Game og In-Game. Dette fordi disse klassene må kunne samhandle uten å lukkes og opprettes hver gang man går til en ny del av spillet. Ytelsesmessig gjør dette at det er litt tyngre å starte, men lettere i bruk siden disse allerede er lastet inn og instansiert. Ved videre utviklingen må bruk av Singletons vurderes nøye, slik at man ikke ender opp med mange unødvendige instanser kjørende.

I hovedsak ble NewGameView den viktigste klassen for å sette sammen alle klassene til et spill som kunne kjøre. Den brukte verdier fra Enum-klasser, spesielt PlayerSpecializationEnum som, kombinert med listeners i comboboxer, satte opp en valgt spillkarakter basert på verdiene tilknyttet de valg man gjorde. Dette ble lagret ved å hente ut og lagre teksten fra disse input-ene og så referere til disse verdiene videre utover i prosessen. Valgene ble lagret i en «GameState»-klasse, som holdt på tilstanden til det valgte spillet, spillerkarakteren, historien man valgte og hvor i historien man var («Passage»). Dette ble også skrevet til en tekstfil.

Game- og Player-klassene fra oppgavespesifikasjonen ble reimplementert i GameState-klassen. Årsaken til dette var i at det virket lettere å implementere den uten å måtte skrive om de eksiterende klassene, og at det i øyeblikket var enklere å skrive den slik at den passet direkte til de behovene som presenterte seg mens NewGameView ble konstruert. I tillegg hadde gruppen tidlig litt problemer med å skjønne hvordan klassene i

oppgaven kunne knyttes sammen og bygges videre, og dermed var det enklere å skrive ny.
Player ble etter hvert tatt i bruk, men Game ble noe overflødig. Det ble gjort forsøk på refaktorisering for å skille Player mer fra GameState og benytte Game, men på grunn av tidsbegrensninger ble det ikke tid til å gjennomføre dette.

En overordnet oversikt over klassene kan ses i vedlagt klassediagram.
Vedlagt er også sekvensdiagrammer for InGameView, MainMenuView, NewGameView og StoryParser.

8 TESTING

Det ble skrevet enhetstester til de grunnleggende klassene i gitt oppgaven. Disse ble skrevet deterministisk, og til å teste både positivt og negativt. Det ble implementert unntakshåndtering i metoder for å forhindre at programmet krasjet som følge av feil.

Det ble utført en enkel brukertest på kode og spill. Foruten innvendinger på det noe uferdige grafiske designet, begrensede muligheter til å sette opp karakteren sin, og lite interaksjon i selve spillet ut over dialogvalg, var tilbakemeldingen i stor grad positiv, situasjonen tatt i betraktning. Det samme gjaldt gjennomgang av kode, hvor brukeren måtte navigere kildekoden for å gjøre endringer vedkommende ønsket. Brukertesten ble gjort veldig uformelt, uten større forsøk på "destruktiv testing". De fleste tilbakemeldingene gjaldt allerede kjente svakheter.

9 UTRULLING TIL SLUTTBRUKER (DEPLOYMENT)

Applikasjonen rulles ut ved at bruker laster ned kildekoden fra GitLab-repoet. Denne kjøres så via et IDE, for eksempel IntelliJ IDEA.

10 DRØFTING

I løpet av prosjektet har teamet fullført et program som lar brukeren velge en historie som skal fortelles, velge karakter og sette mål for spillet gjennom et brukergrensesnitt. Det har blitt lagt mye vekt på å utvikle robust kode, og derfor mangler programmet noen ekstra funksjoner, som lyd, animasjoner, minispill og lagring av fremgang. Årsaken til at disse elementene ikke ble implementert skyldes en undervurdering av arbeidsmengden som var nødvendig for å

utvikle de eksisterende funksjonene. Hvis det samme programmet skulle utvikles igjen, ville det blitt lagt av mer tid til hver iterasjon, slik at mer funksjonalitet kunne blitt legg til.

Flere av de tidligere nevnte funksjonene er påbegynt og har ferdigstilte metoder, men har ikke blitt integrert inn i brukergrensesnittet enda. Dette på grunn av prioriteringer om å forsikre seg om at de funksjonalitetene som er tatt med fungerer og er robuste før mer blir lagt til.

På grunn av kravene til hvordan programmet skulle bli bygget opp i del 1 og 2 hadde teamet litt vansker med å komme i gang og forstå hvordan man skulle videre bygge på de eksisterende klassene. På grunn av dette ble det mye tid som ble kastet bort på å skrive kode som ikke kunne brukes på grunn av at den ikke var kompatibel med hvordan oppdragsgiver ønsket at programmet skulle være oppbygget.

Det er alltids rom for forbedring og med bedre planlegging for bruk av tid ville programmet inneholdt mer funksjonalitet. I sin helhet ble en solid løsning oppnådd gitt tidsbegrensningene og komplikasjonene som ble støtt på underveis. Prosessen bidro til læring ikke bare om tekniske aspekter ved programvareutvikling, men også om prosjektledelse, tidsstyring og teamarbeid.

Kildene som har blitt brukt er flere godt respekterte nettsteder som har blitt brukt av faglærere gjennom emnet. Flere av kildene fungerer som et forum hvor brukere selv kan lage egne poster eller sende inn forbedringsforslag til eksisterende. Dette gjør at kildene er mer anvendelige og troverdige, siden det blir flere individer som kan sjekke og foreslå alternative løsninger, kontra hvis man benytter seg av en kilde som kun er skrevet av én forfatter.

11 KONKLUSJON – ERFARING

Hvis vi ser tilbake på kravspesifikasjonen, er det enkelte mangler om stikker seg ut. Disse går i stor grad ut på manglende implementasjon av Goals på en måte som gjør at de har en innvirkning på spillet. I tillegg gjør mangelen av Actions at spillet fremstår mer som en ikke-lineær visuell novelle, mer enn et spill i den forstand.
I tillegg ble det laget egne klasser i løpet av prosjektets utvikling, som tok over rollene som klasser gitt i oppgaven var ment å ha. Selv om disse klassene gir mye av den samme funksjonaliteten, er ikke disse implementert i henhold til de gitte kravene.

Siden dette prosjektet gikk parallelt med prosjekt i Systemutvikling IDATT1002 som skulle leveres tidligere, ble det prioritert å jobbe der frem til levering. Selv om det ble

gjennomført planleggingsøkter, hvor features og krav ble listet opp og prioritert, burde det blitt viet mer tid til dette, med en tydeligere plan. En skisse av systemet burde blitt laget, for å gjøre arbeidet mer oversiktlig og forutsigbart.

Begrensningene i spillmotoren er i stor grad mangelen av Actions og Goals, som gjør at interaktivitet er noe begrenset. Metodene er implementert, men funksjonaliteten ble ikke fungerende «in-game» i tide til levering. Det uferdige grafisk designet, og *features* som man ser omrisset av, men ikke gjør noe, er også begrensinger som sådan.

I videre utvikling ville prioriteten vært Actions i dialogene. Dette ville gjort spillet mer interaktivt, der handlingene kunne hatt konsekvenser i form av endring i helse, gull, osv. Actions kunne også brukes til turbasert kampmekanikk, slik man kjenner fra japanske rollespill. Dernest ville Goals vært et mål å implementere, for å sette krav som må møtes for å vinne et spill.
Bedre og mer komplett feilhåndtering, og gode tilbakemeldinger på disse i det grafiske grensesnittet, er også høyt prioritert.

Alt i alt er gruppa fornøyd med resultatet. Selv om løsningen ikke ble optimal, ei heller komplett ut fra spesifikasjonene, ble resultatet en klient som kan kjøre en spillfil, og legge til skjermelementer basert på innholdet i fila og hvilke valg som ble gjort i oppsettet til spillet. Altså har prosjektet bidratt til å utvikle gruppas kunnskaper, ikke bare innen programmering, men også hvilke krav som bør stilles til planleggingen av utviklingsprosessen.

12 REFERANSER

[1] Josikakar. (2023, 18. April) Software Engineering | Coupling and Cohesion
https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/

[2] GeeksforGeeks. (2022, 5. Desember) Builder Design Pattern
https://www.geeksforgeeks.org/builder-design-pattern/?ref=gcse

[3] GeeksforGeeks. (2022, 26. Desember) Abstract Factory Pattern
https://www.geeksforgeeks.org/abstract-factory-pattern/?ref=gcse

[4] GeeksforGeeks. (2023, 6. Mars) Java Singleton Class
https://www.geeksforgeeks.org/singleton-class-java/?ref=gcse

[5] Conventional Commits 1.0.0
https://www.conventionalcommits.org/en/v1.0.0/#summary

[6] StackOverflow
https://stackoverflow.com/

[7] Java® Platform, Standard Edition & Java Development Kit
Version 11 API Specification
https://docs.oracle.com/en/java/javase/11/docs/api/index.html

13 VEDLEGG

Link til prosjektets repository:
https://gitlab.stud.idi.ntnu.no/team_12-idatt2001/mappevurdering

Vedlegg til rapporten. Disse kan også finnes på prosjektes Wiki-side:
https://gitlab.stud.idi.ntnu.no/team_12-idatt2001/mappevurdering/-/wikis/home

1. Brukermanual for Paths-formatet.
   Filnavn: *01 - Brukermanual User Manual - Paths format.pdf*
2. Klassediagram Paths.
   Filnavn: *02 - classDiagram-Paths_Full.png*
3. Sekvensdiagram for klassen InGameView.
   Filnavn: *03 - sequenceDiagram-InGameView.png*
4. Sekvensdiagram for klassen MainMenuView.
   Filnavn: *04 - sequenceDiagram-MainMenuView.png*
5. Sekvensdiagram for klassen NewGameView, metode NewGameView.
   Filnavn*: 05 - sequenceDiagram-NewGameView_newGameViewMethod.png*
6. Sekvensdiagram for klassen StoryParser, metode createLinkFromLine.
   Filnavn: *06 - sequenceDiagram-StoryParser_createLinkFromLine.png*
7. Sekvensdiagram for klassen StoryParser, metode parseStoryFromFile.
   Filnavn: *07 - sequenceDiagram-StoryParser_parseStoryFromFile.png*

Paths Story Format

A Story file is written in plain text, but must be written in the following format:

- A passage starts with :: (two colon) and the passage title.
- The next line contains the content of the passage.
- The last line of a block lists the links.
- Each link is on a separate line, on the form [linktext]
- A block is terminated with an empty line.
- **Two empty lines marks the end of the story file.**

Example:

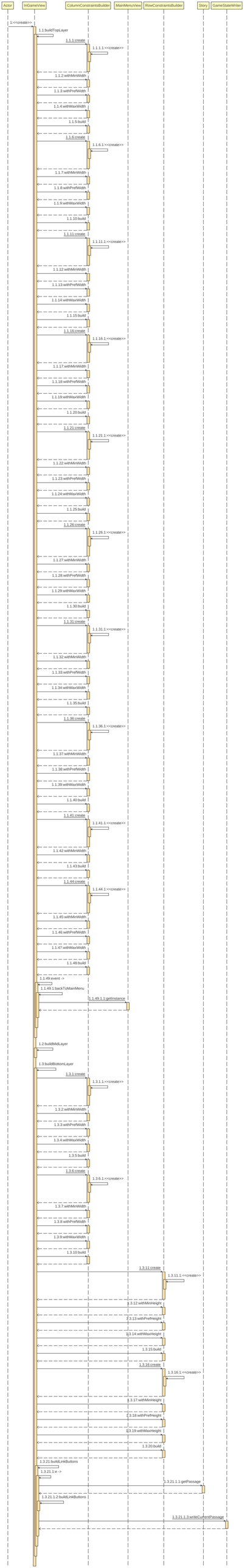
```
Haunted House

::Beginnings
You are in a small, dimly lit room. There is a door in front of you.
[Try to open the door](Another room)

::Another room
The door opens to another room. You see a desk with a large, dusty book.
[Open the book](The book of spells)
[Go back](Beginnings)
```

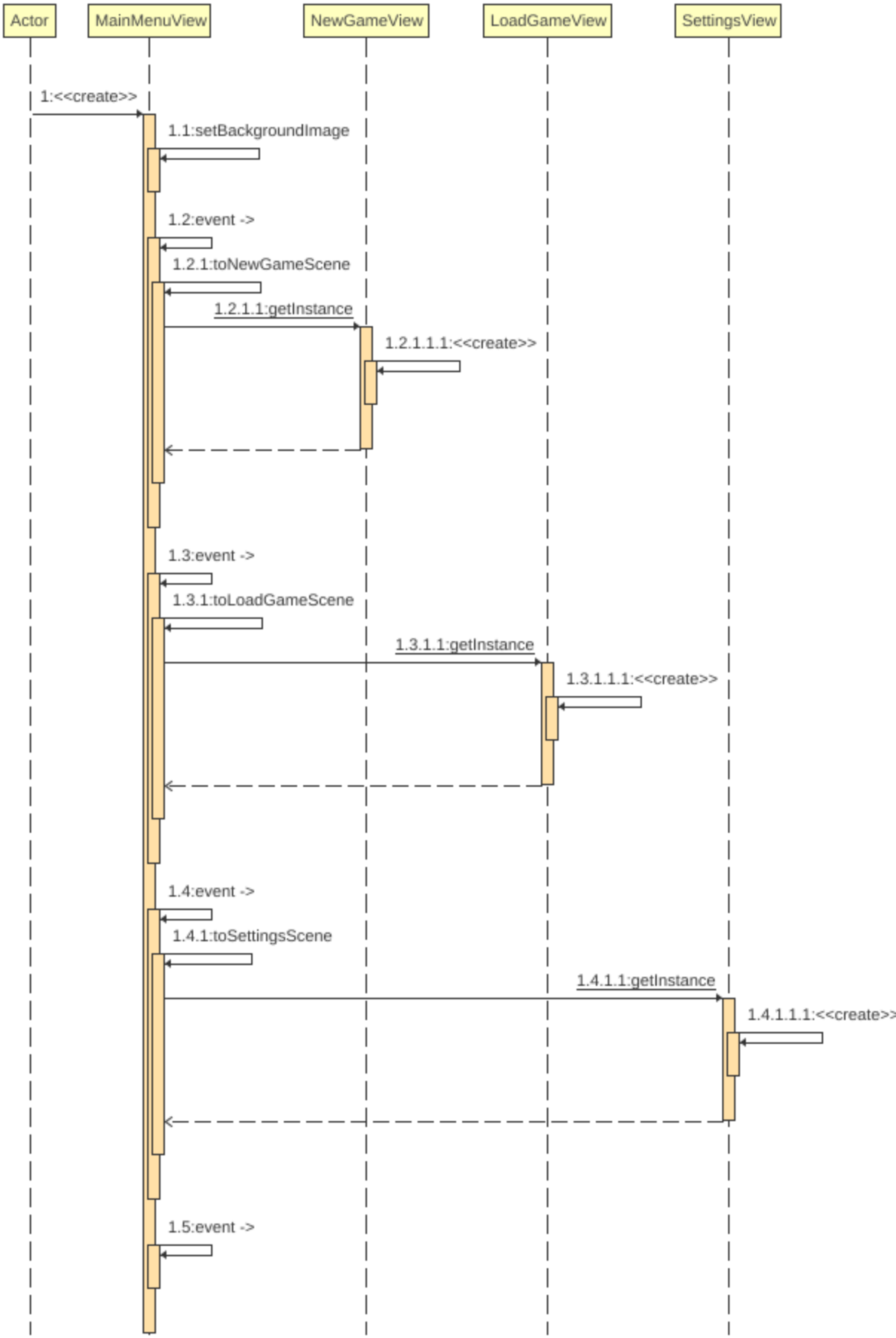
To make it very clearn: the //emptyLine identifies that that line should be empty, and [number] is line numbers.

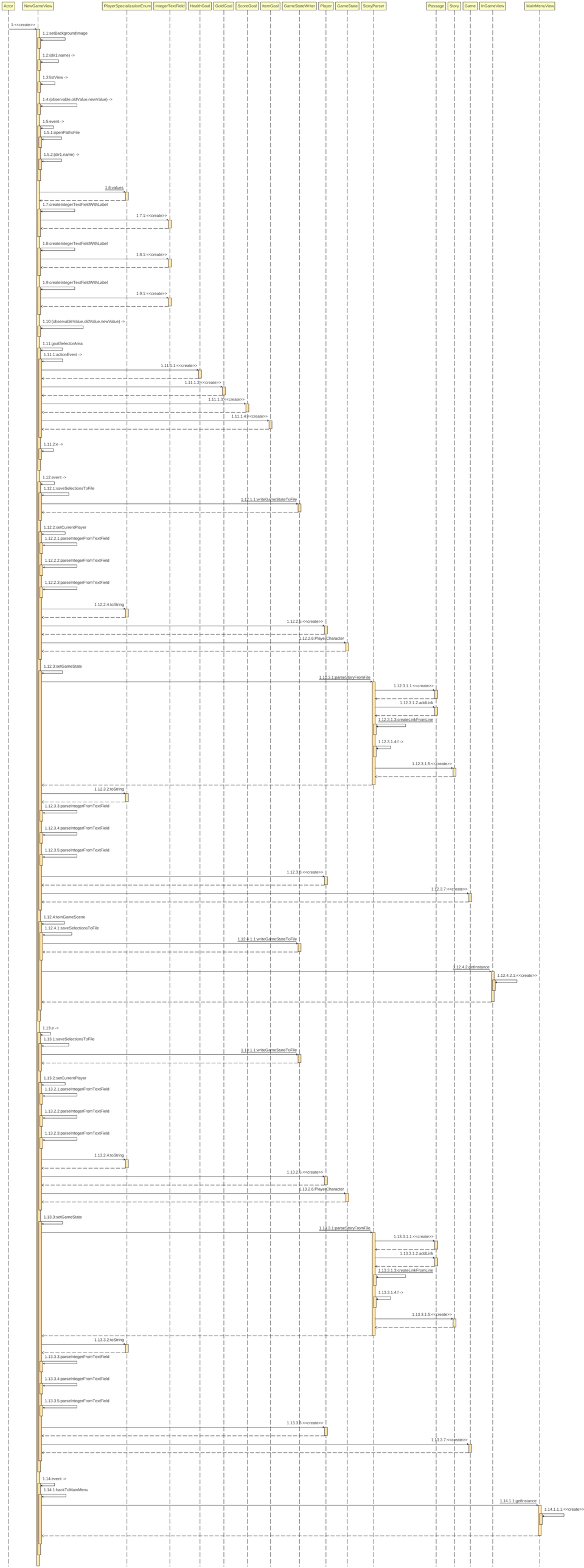
```
[01] Haunted House
[02] //emptyLine
[03] ::Beginnings
[04] You are in a small, dimly lit room. There is a door in front of you.
[05] [Try to open the door](Another room)
[06] //emptyLine
[07] ::Another room
[08] The door opens to another room. You see a desk with a large, dusty book.
[09] [Open the book](The book of spells)
[10] [Go back](Beginnings)
[11] //emptyLine
[12] //emptyLine
```

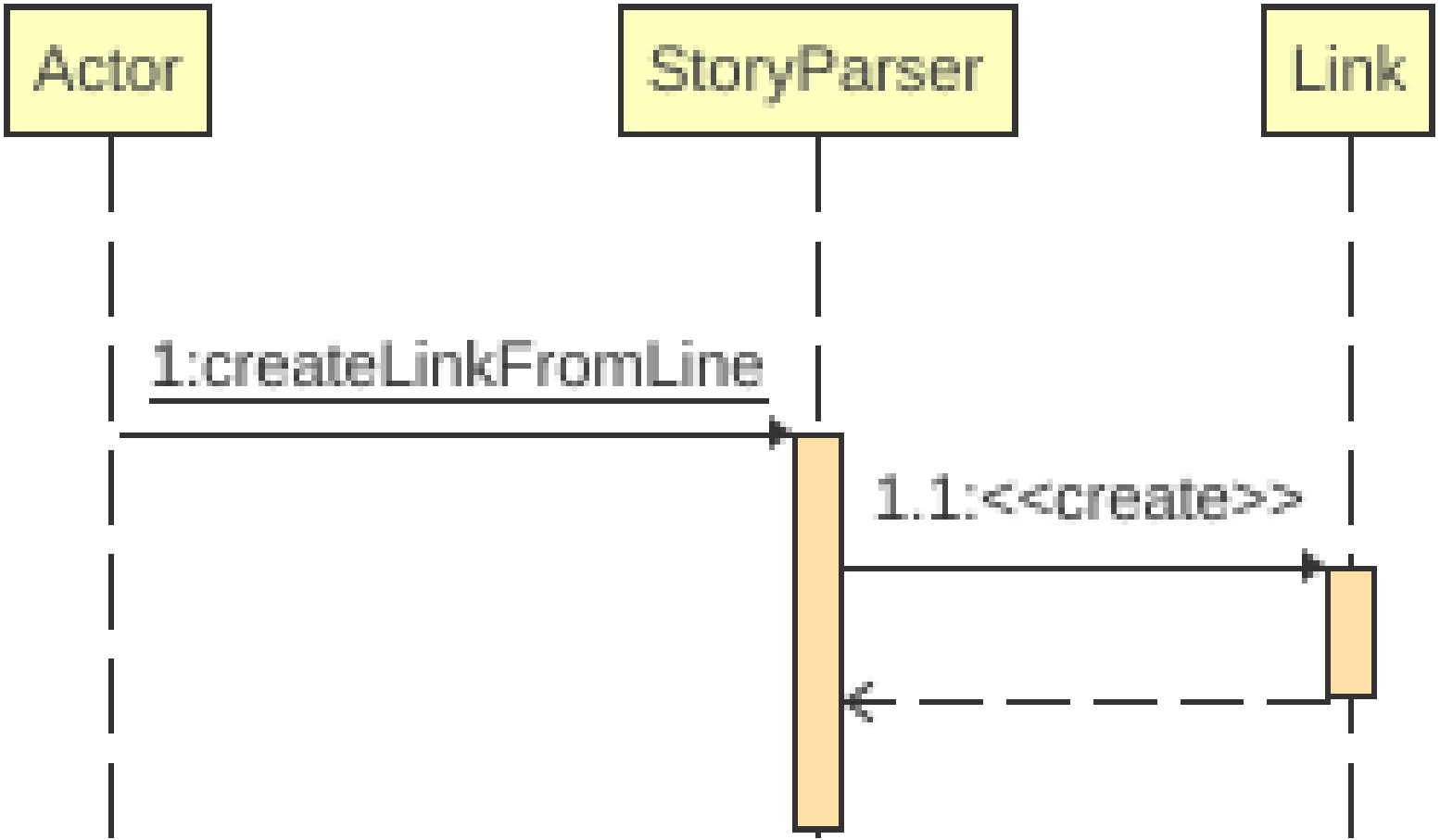
The file must have the file ending `.paths`, e.g. The `Best Story Ever.paths` It must be placed in the project directory `story/`, found at `src/main/resources/story/`

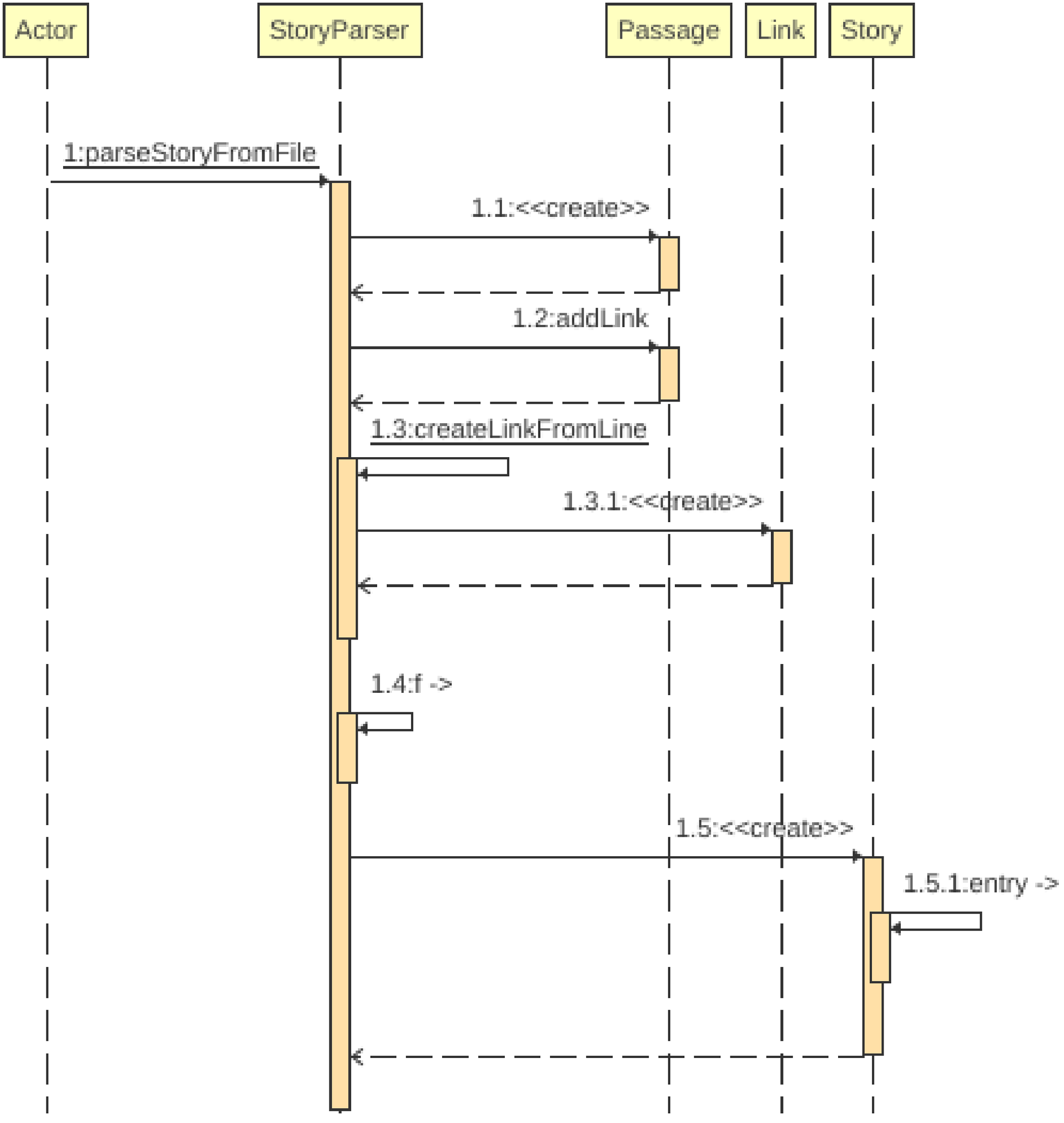

Player
Player(String, int, int, int, List<String>, String, Image)
inventory List<String>
currentLink Link
health int
score int
playerSpecializationImage Image
playerSpecialization String
name String
gold int
toString() String
addScore(int) int
removeFromInventory(String) void
setCharacter() void
addGold(int) int
removeItemFromInventory(String) void
getCharacter() void
hasItem(String) boolean
attack(int) int
addHealth(int) int
useWeapon(String) void
addToInventory(String) void
hasItemInInventory(String) boolean
name String
gold int
playerSpecialization String
inventory List<String>
score int
health int
playerSpecializationImage Image
currentLink Object
GameState
GameState()
startingScore int
passage Passage
playerName String
startingHP int
startingItem String
playerCharacterImage Image
startingItemIcon String
player Player
startingGold int
selectedStoryFile File
PlayerCharacter(Player) void
startingScore int
playerCharacterImage Image
passage Passage
playerName String
startingHP int
startingItem String
playerSpecialization PlayerSpecializationEnum
startingGold int
player Player
instance GameState
selectedStoryFile File
startingItemIcon String
Action
execute(Player) void
GoldAction
GoldAction(int)
hashCode() int
execute(Player) void
HealthAction
HealthAction(int)
execute(Player) void
hashCode() int
Link
Link(String, String)
text String
reference String
actions List<Action>
hashCode() int
addAction(List<Action>) void
equals(Object) boolean
execute(Player) void
toString() String
reference String
text String
actions List<Action>
ScoreAction
ScoreAction(int)
hashCode() int
execute(Player) void
InventoryAction
InventoryAction(String, boolean)
InventoryAction(String)
execute(Player) void
NewGameView
NewGameView(Stage)
sceneNewGame Scene
setBackgroundImage() void
setGameState() void
getInstance(Stage) NewGameView
backToMainMenu() void
createIntegerTextFieldWithLabe(Label) TextField
parseIntegerFromTextField(TextField, int) int
toInGameScene() void
setCurrentPlayer() void
goalSelectorArea() VBox
openPathsFile() void
saveSelectionsToFile() void
stageNewGame Stage
sceneNewGame Scene
ItemsEquipmentAndArmor
ItemsEquipmentAndArmor(String, int, int, int, String)
baseDefence int
itemName String
filePath String
bonusMitigationChance int
magicDefence int
toString() String
values() ItemsEquipmentAndArmor[]
valueOf(String) ItemsEquipmentAndArmor
itemName String
filePath String
bonusMitigationChance int
magicDefence int
baseDefence int
ItemsPotionsEnum
ItemsPotionsEnum(String, int, int, int, String)
manaAdded int
protectionAdded int
healthAdded int
filePath String
itemName String
toString() String
values() ItemsPotionsEnum[]
valueOf(String) ItemsPotionsEnum
itemName String
filePath String
healthAdded int
protectionAdded int
manaAdded int
Game
Game(Player, Story, List<Goal>)
goals List<Goal>
story Story
player Player
saveGame() void
gameLost() boolean
go(Link) Passage
gameWon() boolean
begin() Passage
resumeGame() void
checkGameGoals() Map<String, Boolean>
player Player
story Story
goals List<Goal>
PlayerBuilder
PlayerBuilder(Builder)
getGold(int) int
getInventory(List<String>) List<String>
getHealth(int) int
getScore(int) int
getName(String) String
PlayerSpecializationEnum
PlayerSpecializationEnum(String, String, String, String)
imagePath String
startingItem String
startingItemIcon String
displayName String
toString() String
valueOf(String) PlayerSpecializationEnum
values() PlayerSpecializationEnum[]
startingItem String
displayName String
imagePath String
startingItemIcon String
ItemsWeaponsEnum
ItemsWeaponsEnum(String, int, int, int, String)
criticalDamage int
baseDamage int
itemName String
filePath String
values() ItemsWeaponsEnum[]
valueOf(String) ItemsWeaponsEnum
toString() String
itemName String
filePath String
criticalDamage int
baseDamage int
Passage
Passage(String, String)
title String
links List<Link>
content String
hasLinks() boolean
equals(Object) boolean
hashCode() int
addLink(Link) boolean
toString() String
content String
links List<Link>
title String
Story
Story(String, Map<Link, Passage>)
openingPassage Passage
passages Map<Link, Passage>
title String
addPassage(Passage) boolean
getPassage(Link) Passage
removePassage(Link) void
passages Map<Link, Passage>
title String
openingPassage Passage
brokenLinks List<Link>
MainView
MainView()
scene Scene
inGameView InGameView
loadGameView LoadGameView
newGameView NewGameView
main(String[]) void
start(Stage) void
scene Scene
newGameView NewGameView
loadGameView LoadGameView
inGameView InGameView
StoryFileTitleReader
StoryFileTitleReader(File)
file File
title StringProperty
description StringProperty
titleProperty() StringProperty
descriptionProperty() StringProperty
getFilesFromDirectory(String) List<StoryFileTitleReader>
file File
description String
title String
NonPlayerCharacterEnum
NonPlayerCharacterEnum(String, String, String)
filePath String
itemLoot String
displayName String
valueOf(String) NonPlayerCharacterEnum
toString() String
values() NonPlayerCharacterEnum[]
displayName String
filePath String
itemLoot String
ItemsMiscellaneousEnum
ItemsMiscellaneousEnum(String, String, String)
itemDescription String
itemName String
filePath String
valueOf(String) ItemsMiscellaneousEnum
toString() String
values() ItemsMiscellaneousEnum[]
itemName String
filePath String
itemDescription String
MainMenuView
MainMenuView(Stage)
sceneMainMenu Scene
getInstance(Stage) MainMenuView
toNewGameScene() void
setBackgroundImage() void
toSettingsScene() void
toLoadGameScene() void
stageMainMenu Stage
sceneMainMenu Scene
InGameView
InGameView(Stage)
buildTopLayer() GridPane
buildMidLayer() StackPane
backToMainMenu() void
getInstance(Stage) InGameView
buildLinkButtons(Passage) ArrayList<Button>
buildBottomLayer() HBox
sceneInGame Scene
stageInGame Stage
Goal
isFulfilled(Player) boolean
GoldGoal
GoldGoal(int)
isFulfilled(Player) boolean
toString() String
value int
InventoryGoal
InventoryGoal(List<String>)
isFulfilled(Player) boolean
toString() String
items List<String>
ScoreGoal
ScoreGoal(int)
toString() String
isFulfilled(Player) boolean
value int
HealthGoal
HealthGoal(int)
isFulfilled(Player) boolean
toString() String
value int
ItemGoal
ItemGoal(String)
isFulfilled(Player) boolean
toString() String
LoadGameView
LoadGameView(Stage)
sceneLoadGame Scene
setBackgroundImage() void
backToMainMenu() void
getInstance(Stage) LoadGameView
stageLoadGame Stage
sceneLoadGame Scene
SettingsView
SettingsView(Stage)
sceneSettings Scene
backToMainMenu() void
getInstance(Stage) SettingsView
sceneSettings Scene
stageSettings Stage
ColumnConstraintsBuilder
ColumnConstraintsBuilder()
build() ColumnConstraints
create() ColumnConstraintsBuilder
withPrefWidth(double) ColumnConstraintsBuilder
withMinWidth(double) ColumnConstraintsBuilder
withMaxWidth(double) ColumnConstraintsBuilder
RowConstraintsBuilder
RowConstraintsBuilder()
withMaxHeight(double) RowConstraintsBuilder
create() RowConstraintsBuilder
build() RowConstraints
withMinHeight(double) RowConstraintsBuilder
withPrefHeight(double) RowConstraintsBuilder
GameStateWriter
GameStateWriter()
writeGameStateToFile(PlayerSpecializationEnum, File, String,)
writeCurrentPassage(String) void
StoryParser
StoryParser()
createLinkFromLine(String) Link
parseStoryFromFile(String) Story?
StoryFileHandler
StoryFileHandler()
saveStory(Story, String) void
loadStory(String) Story
Main
Main()
main(String[]) void
IntegerTextField
IntegerTextField(Label)
InGameMenuView
InGameMenuView()
Constants
Constants()
Builder
Builder(String, int)
score(int) Builder
gold(int) Builder
inventory(List<String>) Builder


Actor
InGameView
ColumnConstraintsBuilder
MainMenuView
RowConstraintsBuilder
Story
GameStateWriter
1:<<create>>
1.1:buildTopLayer
1.1.1:create
1.1.1.1:<<create>>
1.1.2:withMinWidth
1.1.3:withPrefWidth
1.1.4:withMaxWidth
1.1.5:build
1.1.6:create
1.1.6.1:<<create>>
1.1.7:withMinWidth
1.1.8:withPrefWidth
1.1.9:withMaxWidth
1.1.10:build
1.1.11:create
1.1.11.1:<<create>>
1.1.12:withMinWidth
1.1.13:withPrefWidth
1.1.14:withMaxWidth
1.1.15:build
1.1.16:create
1.1.16.1:<<create>>
1.1.17:withMinWidth
1.1.18:withPrefWidth
1.1.19:withMaxWidth
1.1.20:build
1.1.21:create
1.1.21.1:<<create>>
1.1.22:withMinWidth
1.1.23:withPrefWidth
1.1.24:withMaxWidth
1.1.25:build
1.1.26:create
1.1.26.1:<<create>>
1.1.27:withMinWidth
1.1.28:withPrefWidth
1.1.29:withMaxWidth
1.1.30:build
1.1.31:create
1.1.31.1:<<create>>
1.1.32:withMinWidth
1.1.33:withPrefWidth
1.1.34:withMaxWidth
1.1.35:build
1.1.36:create
1.1.36.1:<<create>>
1.1.37:withMinWidth
1.1.38:withPrefWidth
1.1.39:withMaxWidth
1.1.40:build
1.1.41:create
1.1.41.1:<<create>>
1.1.42:withMinWidth
1.1.43:build
1.1.44:create
1.1.44.1:<<create>>
1.1.45:withMinWidth
1.1.46:withPrefWidth
1.1.47:withMaxWidth
1.1.48:build
1.1.49:event ->
1.1.49.1:backToMainMenu
1.1.49.1.1:getInstance
1.2:buildMidLayer
1.3:buildBottomLayer
1.3.1:create
1.3.1.1:<<create>>
1.3.2:withMinWidth
1.3.3:withPrefWidth
1.3.4:withMaxWidth
1.3.5:build
1.3.6:create
1.3.6.1:<<create>>
1.3.7:withMinWidth
1.3.8:withPrefWidth
1.3.9:withMaxWidth
1.3.10:build
1.3.11:create
1.3.11.1:<<create>>
1.3.12:withMinHeight
1.3.13:withPrefHeight
1.3.14:withMaxHeight
1.3.15:build
1.3.16:create
1.3.16.1:<<create>>
1.3.17:withMinHeight
1.3.18:withPrefHeight
1.3.19:withMaxHeight
1.3.20:build
1.3.21:buildLinkButtons
1.3.21.1:e ->
1.3.21.1.1:getPassage
1.3.21.1.2:buildLinkButtons
1.3.21.1.3:writeCurrentPassage


Actor
MainMenuView
NewGameView
LoadGameView
SettingsView
1:<<create>>
1.1:setBackgroundImage
1.2:event ->
1.2.1:toNewGameScene
1.2.1.1:getInstance
1.2.1.1.1:<<create>>
1.3:event ->
1.3.1:toLoadGameScene
1.3.1.1:getInstance
1.3.1.1.1:<<create>>
1.4:event ->
1.4.1:toSettingsScene
1.4.1.1:getInstance
1.4.1.1.1:<<create>>
1.5:event ->


Actor
NewGameView
PlayerSpecializationEnum
IntegerTextField
HealthGoal
GoldGoal
ScoreGoal
ItemGoal
GameStateWriter
Player
GameState
StoryParser
Passage
Story
Game
InGameView
MainMenuView
1:<<create>>
1.1:setBackgroundImage
1.2:(dir1,name) ->
1.3:listView ->
1.4:(observable,oldValue,newValue) ->
1.5:event ->
1.5.1:openPathsFile
1.5.2:(dir1,name) ->
1.6:values
1.7:createIntegerTextFieldWithLabel
1.7.1:<<create>>
1.8:createIntegerTextFieldWithLabel
1.8.1:<<create>>
1.9:createIntegerTextFieldWithLabel
1.9.1:<<create>>
1.10:(observableValue,oldValue,newValue) ->
1.11:goalSelectorArea
1.11.1:actionEvent ->
1.11.1.1:<<create>>
1.11.1.2:<<create>>
1.11.1.3:<<create>>
1.11.1.4:<<create>>
1.11.2:e ->
1.12:event ->
1.12.1:saveSelectionsToFile
1.12.1.1:writeGameStateToFile
1.12.2:setCurrentPlayer
1.12.2.1:parseIntegerFromTextField
1.12.2.2:parseIntegerFromTextField
1.12.2.3:parseIntegerFromTextField
1.12.2.4:toString
1.12.2.5:<<create>>
1.12.2.6:PlayerCharacter
1.12.3:setGameState
1.12.3.1:parseStoryFromFile
1.12.3.1.1:<<create>>
1.12.3.1.2:addLink
1.12.3.1.3:createLinkFromLine
1.12.3.1.4:f ->
1.12.3.1.5:<<create>>
1.12.3.2:toString
1.12.3.3:parseIntegerFromTextField
1.12.3.4:parseIntegerFromTextField
1.12.3.5:parseIntegerFromTextField
1.12.3.6:<<create>>
1.12.3.7:<<create>>
1.12.4:toInGameScene
1.12.4.1:saveSelectionsToFile
1.12.4.1.1:writeGameStateToFile
1.12.4.2:getInstance
1.12.4.2.1:<<create>>
1.13:e ->
1.13.1:saveSelectionsToFile
1.13.1.1:writeGameStateToFile
1.13.2:setCurrentPlayer
1.13.2.1:parseIntegerFromTextField
1.13.2.2:parseIntegerFromTextField
1.13.2.3:parseIntegerFromTextField
1.13.2.4:toString
1.13.2.5:<<create>>
1.13.2.6:PlayerCharacter
1.13.3:setGameState
1.13.3.1:parseStoryFromFile
1.13.3.1.1:<<create>>
1.13.3.1.2:addLink
1.13.3.1.3:createLinkFromLine
1.13.3.1.4:f ->
1.13.3.1.5:<<create>>
1.13.3.2:toString
1.13.3.3:parseIntegerFromTextField
1.13.3.4:parseIntegerFromTextField
1.13.3.5:parseIntegerFromTextField
1.13.3.6:<<create>>
1.13.3.7:<<create>>
1.14:event ->
1.14.1:backToMainMenu
1.14.1.1:getInstance
1.14.1.1.1:<<create>>


Actor
StoryParser
Link
1:createLinkFromLine
1.1:<<create>>


Actor
StoryParser
Passage
Link
Story
1:parseStoryFromFile
1.1:<<create>>
1.2:addLink
1.3:createLinkFromLine
1.3.1:<<create>>
1.4:f ->
1.5:<<create>>
1.5.1:entry ->
1.5.1:entry ->