eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

Facultad de Informática

Grado de Ingeniería Informática

▪ Proyecto Fin de Grado ▪

Ingeniería de Computadores

Interactive and collaborative creation of stories

Creación interactiva y colaborativa de historias

Olatz Castaño

Septiembre 2016

informatika fakultatea
facultad de informática

Agradecimientos

Antes de nada, me gustaría agradecer a mi supervisor, Alejandro García-Alonso, la completa confianza que ha depositado en mí para realizar este proyecto. Sin ella, y sin la rápida asistencia que me has ofrecido siempre que ha habido algún problema, no habría sido posible llevarlo a cabo. Gracias.

También me gustaría agradecer a Rafael Bidarra y a Ben Kybartas la oportunidad que me han concedido de trabajar con ellos en este proyecto y tenerme en TU Delft. La experiencia que he ganado trabajando con vosotros es invaluable.

Por último, pero no menos importante, quiero agradecer a mi madre, Kontxi Pérez, por apoyarme y darme la oportunidad de perseguir mis sueños, manteniéndome mientras estaba fuera trabajando en este proyecto. Todo ha sido posible gracias a ti.

Resumen

Esta memoria contiene la definición de una estructura capaz de guardar historias de una manera comprensible y coherente, que facilitará la investigación y el desarrollo de software. También contiene la implementación de una aplicación, persiguiendo dicho objetivo, TaleBox, que ha sido desarrollado como un juego de cartas por turnos y multijugador para dispositivos móviles, para la creación colaborativa e interactiva de historias.

Ambas, estructura y aplicación, han sido desarrolladas con la ayuda de Ben Kybartas y GluNet, y bajo supervisión directa de Rafael Bidarra en la TU Delft, Delft, Países Bajos.

Índice

Lista de Figuras y Tablas

FIGURAS

TABLAS

ALGORITMOS

1

Introducción

1.1. Objetivos

Este proyecto ha sido realizado en la TU Delft, Delft, Países Bajos, bajo la supervisión de Rafael Bidarra y con la colaboración de Ben Kybartas.

El objetivo global era proporcionar las herramientas necesarias para la creación y el intercambio de historias *Open Source* a la comunidad de investigación y desarrollo de software.

Para ello, se dividió el proyecto en dos objetivos menores, cuya unión haría posible el primero.

1.1.1. StoryDB

Se requiere de una estructura capaz de guardar historias de forma coherente y legible tanto para un humano como para una máquina. Esto permitirá compartirlas y las hará útiles para desarrolladores e investigadores. Esta estructura será llamada StoryDB.

Apoyándose en las investigaciones sobre análisis de historias ya existentes, StoryDB deberá mantener la coherencia entre las distintas partes de la historia y aportar algo nuevo a la comunidad científica.

1.1.2. TaleBox

TaleBox será la herramienta que sirva para introducir historias en StoryDB. Esta herramienta debe ser capaz de ofrecer un amplio abanico de posibilidades a los usuarios sin poner en peligro la coherencia de la historia.

Ya que se quiere atraer a un gran público para utilizar la herramienta, de forma que StoryDB contenga muchas historias originales y distintas, se ha decidido darle un objetivo secundario: TaleBox será un videojuego. Este videojuego permitirá a los jugadores crear historias de forma creativa y colaborativa, y compartirlas con sus amistades. Debe ser atrayente y fácil de usar, para que los jugadores no pierdan el interés, sin perder su gran flexibilidad ni restringir la creatividad.

2

Investigación Previa

En este capítulo se ofrece un rápido análisis de lo descubierto en la investigación previa a enfrentar los objetivos planteados para el proyecto, y presentar la motivación detrás de las decisiones tomadas durante el desarrollo del mismo.

2.1. Análisis de Historias

La construcción de una estructura como StoryDB requiere de una comprensión precisa de la forma de una historia, así como del análisis de su estructura y sus partes. Es necesario, por tanto, fijarse en el trabajo que se ha hecho previamente respecto al análisis de historias. Los artículos elegidos para esta parte de la investigación previa han sido aquellos que intentan desgranar, desde un punto de vista computacional, las fórmulas para analizar historias y crear esquemas a partir de éstas, esquemas que puedan utilizar los computadores sin pasar por el análisis lingüístico del texto.

Para ello, se ha estudiado la investigación de David K. Elson[1][2] y su construcción de los Grafos de Intención de las Historias (Story Intention Graph – SIG).

2.1.1. Story Intention Graph (SIG)

El objetivo del SIG es determinar una metodología para la investigación de la estructura narrativa en base a los datos.

Divide el texto en tres capas: textual, cronológica e interpretativa.

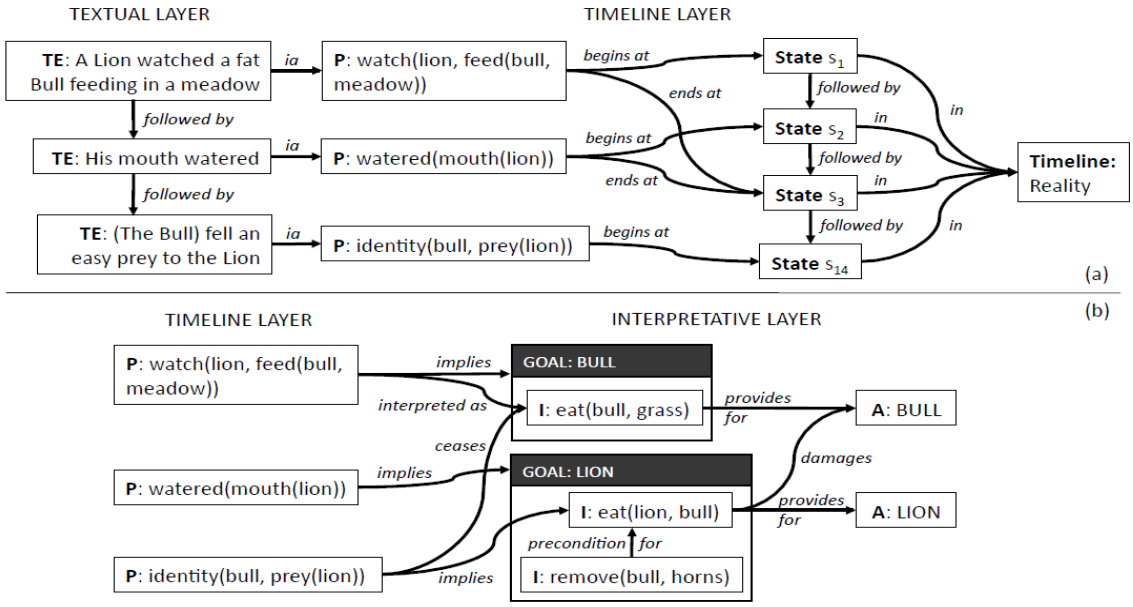


Figura 2.1 SIG de «The Wily Lion», extraída del artículo *DramaBank: Annotating Agency in Narrative Discourse*, por David K. Elson.

En esta figura, extraída del artículo *DramaBank: Annotating Agency in Narrative Discourse*, podemos observar el resultado de la construcción de un SIG. Los SIG son construidos de forma manual por agentes humanos, utilizando un software especial para ello y una lista de

indicaciones. Nuestro objetivo, sin embargo, es guardar la información necesaria para que un computador pueda derivar un SIG de una historia.

El SIG observa el texto y deriva acciones de él, que convierte en nodos. Estos nodos se relacionan en eventos, *estados* de la historia, definiendo en la capa cronológica dónde empiezan y acaban. De las acciones, siguiendo siempre la notación particular de este método, extrae *metas*, objetivos, para los personajes y elementos de la historia, y los coloca en la capa interpretativa. A estas metas se les añaden precondiciones y obstáculos que, en conjunto, muestran un complejo cuadro con todos los datos importantes de la historia en su aspecto narrativo, siempre desde el punto de vista del lector.

Podríamos resumirlo en que de la capa de texto extrae nodos de acción, que luego relaciona en *eventos* dentro de la capa cronólogica. Estos nodos evento, a su vez, se relacionan con los nodos de la capa interpretativa, que nos indican la interpretación del lector de las intenciones de los personajes dentro de la historia. Por último, se relacionan los nodos de acción con las intenciones de los personajes, ofreciendo un cuadro que las intenciones dentro de la historia; sus metas, precondiciones y obstáculos.

Observando la metodología del SIG, podemos denotar todos los puntos importantes a estudiar de la narrativa, y que es nuestro objetivo contener en StoryDB. Necesitamos estudiar el espacio de la historia (los personajes, los objetos, etc.), las metas y objetivos de sus elementos y la correlación entre los eventos que se van sucediendo, para poder guardar una historia de forma coherente, legible y estructurada.

2.1.2. Análisis de Caperucita Roja y La Hermana Zorro

Siguiendo el modelo planteado por el SIG, se realizó un experimento donde se analizaban dos historias, Caperucita Roja y La Hermana Zorro, bajo la perspectiva del SIG y nuestra aproximación a StoryDB.

Este paso fue intermedio entre la investigación previa al proyecto y el diseño de StoryDB, proveyéndonos de una visión global del problema antes de desmenuzarlo para el estudio de las necesidades de la estructura objetivo. Debido a la gran extensión de ambos análisis, estos se encuentran como anexos a la memoria.

2.2. Herramientas narrativas

A la hora de abordar TaleBox, hay que tener en cuenta las muchas otras aplicaciones existentes en este ámbito. Podemos tomar como ejemplo el trabajo de David K. Elson y Kathleen R. McKeown[3][4] con su Banco de Narrativas Semánticamente Codificadas y la herramienta que desarrollaron: Scheherezade, o el trabajo de Rafael Bidarra y Ben Kybartas[5] en TU Delft, con los juegos Smart Storybook e Improv Game.

La aproximación a la funcionalidad deseada de estas herramientas suele ser ofrecer al usuario una gran cantidad de información o una serie de palabras aleatorias con las que formar eventos para sus historias. Pero esto no explora la creatividad del autor ni tiene en cuenta la coherencia

de la historia, sino que le presenta con una serie de opciones de las que debe elegir; o una lista inmensa de palabras o una serie de oraciones precocinadas que, como resultado, dan historias divertidas pero incoherentes. En el caso más flexible de software para la creación de historias, tenemos Scrivener[6], un procesador de texto construido con el fin de ayudar al autor a *escribir* la historia.

Nuestro objetivo es beneficiarnos de GluNet[7], una herramienta desarrollada y aportada por Ben Kybartas a este proyecto, para ofrecer un punto medio al usuario. En vez de ofrecerle una base de datos lingüística completa con las herramientas para manejarla, o de pre-seleccionar una serie de frases hechas cuyo marco es estricto y sólido, se quiere preseleccionar un conjunto de palabras coherente con la historia escrita hasta el momento, cuya combinación se deje completamente a la creatividad y alberdrío del autor. Esto, en esencia, le dará al usuario la capacidad de crear sus propias historias a partir de un conjunto controlado, pero amplio y flexible, sin que esto cohíba la coherencia de la historia o su creatividad.

3

Teoría de la Construcción de Historias

Antes de diseñar StoryDB es necesario comprender y analizar de forma precisa todas la partes de una historia, la composición de sus elementos y lo que le da coherencia. Para representar historias con el mayor número de datos e información posible, se necesita saber los elementos más importantes que la componen.

En este capítulo realizamos una tarea de ingeniería inversa para entender la construcción de una historia desde el principio.

3.1. Estructura en Tres Actos

Tradicionalmente, una historia se puede dividir en tres períodos: la introducción, el nudo y el desenlace. Esta descripción corresponde a la estructura argumental, una versión simplificada de la estructura en tres actos, que veremos a continuación:

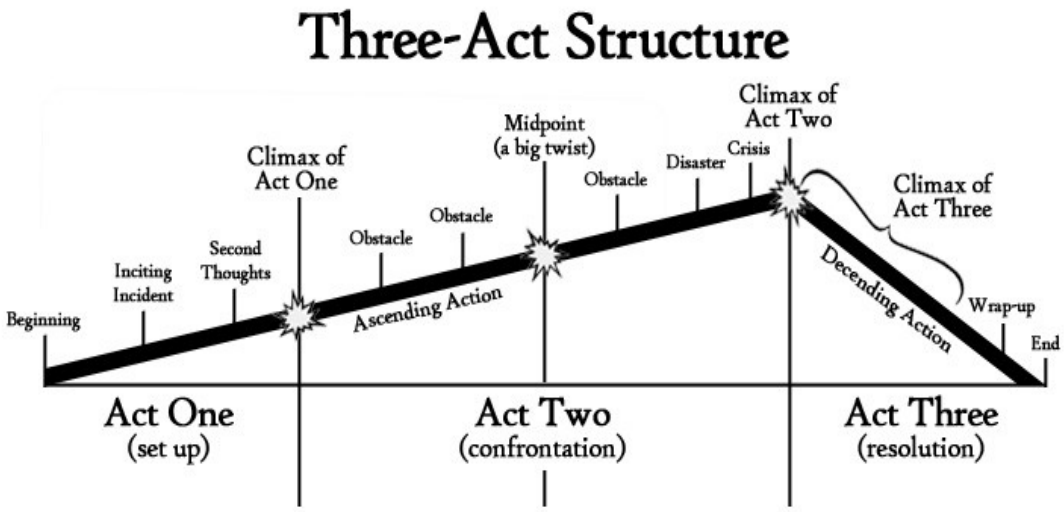


Figura 3.1 Estructura en tres actos (*http://www.raindance.org*).

Bajo este esquema se pueden apreciar los puntos clave por los que pasa una historia en su transcurso. Si bien no todas las historias lo han seguido al ser construidas, la mayoría pueden ser divididas con este esquema. La línea resaltada indica la tensión ascendente que debe llevar la trama.

Cabe destacar que aunque una historia sea construida con este esquema, ello no implica que después sea presentada al lector o al espectador en el mismo orden. Es decir, se podría presentar primero el acto tercero, luego el primero y por último el segundo. El objetivo del esquema es relacionar cronológicamente los eventos a fin de ayudar al autor a construir una historia coherente.

3.1.1. Primer Acto

En el Primer Acto se da la Introducción a la historia.

Empieza en el Status Quo, el momento donde se presentan las bases del mundo en el que estamos a punto de entrar, quién es el protagonista, cuál es su vida diaria... La amplitud del

Status Quo puede variar y su función única es la de dar un punto de partida al lector; no implica presentar a todos los personajes.

El inicio de la historia como tal se da en el **Detonante** (*Inciting Incident* o *Trigger*), el evento que pone en marcha los eventos de la historia. El Detonante suele cambiar, de alguna forma, lo presentado en el Status Quo. Cambia la vida normal del protagonista para incitarle a salir de ese marco y perseguir la historia. Se presenta un conflicto y el protagonista adquiere como meta resolverlo. Esta meta perdurará hasta la resolución de la historia, donde se descubrirá si se alcanza o no.

Tras esto y una serie de eventos que puedan darse, llega el **Primer Punto de Giro** (*Second Thoughts* o *First Pivot Point*). El protagonista descubre un rumbo hacia la resolución del conflicto y se pone en camino, dando paso al Segundo Acto.

3.1.2. Segundo Acto

En el Segundo Acto, como Nudo de la historia, se dan la mayor parte de sucesos de la trama.

Nos encontramos la sucesión de obstáculos hacia el enfrentamiento final (el Climax), que añaden tensión a la historia de forma paulatina y constante. Es también, por costumbre, el acto más grande de los tres, comprendiendo desde que el protagonista se embarca hasta poco antes de su final.

En todo el cuerpo del Segundo Acto hay sólo dos paradas:

La primera, el **Punto Medio** (*Mid point [a big twist]*), se trata de un evento opcional, y más o menos en la mitad de la historia, que cambiará o desviará el objetivo principal del protagonista.

La segunda, casi al final del Segundo Acto, es el **Segundo Punto de Giro** (*Crisis* o *Second Pivot Point*). Se trata de un falso climax que lleva al tercer y último acto. Se puede denominar también como un primer enfrentamiento con el conflicto que no acaba por resolverse, o el punto medio de un evento menor, cuyo desarrollo que empieza en el Desastre, alcanza su pico en la Crisis (Segundo Punto de Giro) y acaba en el Climax de la historia, ya en el Tercer Acto.

3.1.3. Tercer Acto

El Tercer Acto es el último de una historia y corresponde al Desenlance. Se resuelven todos los conflictos, subtramas y siembras que hayan aparecido durante la historia y aún no hayan tenido un final o una cosecha. El Tercer Acto suele ser el menor en longitud, ya que su objetivo es resolver la tensión con rapidez y llevar la historia a su final.

El **Climax** ocurre poco después de entrar en el Tercer Acto. Es el enfrentamiento final del protagonista con el conflicto principal de la historia, que se resuelve, sea de forma favorable o no. El Climax es el pico de tensión de la historia, que ha ido creciendo obstáculo a obstáculo durante el Segundo Acto. Tras el Climax, la tensión desciende en picado.

El **Final** (*Ending)* es el estado final de la historia. Podría tratarse como un Epílogo, donde se informa al lector o al espectador de cómo quedaron las cosas tras los eventos dados. Es el equivalente al Status Quo, ya que su única función es ilustrar el espacio de la historia tras los conflictos ocurridos en ella.

3.1.4. Subtramas y Siembra-Cosechas

Las **subtramas** ocurren junto a los eventos como pequeñas historias enganchadas a la principal. Su longitud no suele (o no debe) exceder los dos actos, pero su esquema es el mismo que el de la historia principal. Es decir, una subtrama es construida con la estructura de tres actos, al igual que cualquier historia, pero teniendo en cuenta que debe ser conexa con la trama principal.

Por otro lado, se denominan **siembras y cosechas** a pequeños detalles o pistas que se le dan al lector o espectador durante la historia para ser «cosechados» más adelante. Es decir, objetos u elementos de la trama que jugarán algún papel en la resolución de conflictos u obstáculos que los personajes puedan encontrarse.

3.2. Temas

El Tema es el tópico o el asunto que se va a tratar en la historia. Aunque una obra puede tratar muchos temas y entablar muchísimas discusiones, se define como Tema al tópico núcleo que guía la historia. Éste puede ser cualquier cosa de la que el autor quiera hablar; la muerte de alguien, el dolor, el amor, el sacrificio, la amistad...

Dado que para entender y determinar con precisión el tema de una obra es necesario analizar su texto, cosa que no está en los objetivos de este proyecto, se ha decidido guardar el Tema como información descriptiva de la obra en la estructura.

3.3. Tramas Maestras

Llamamos Trama a la estructura de eventos que sigue una historia para enfrentar o resolver un conflicto. Con frecuencia, cuando varias historias comparten un mismo concepto en el conflicto a resolver, la serie de eventos y obstáculos que se presentan al/los protagonista(s) suelen tener similitudes. Durante años se han investigado a fondo las miles de historias escritas por la humanidad para localizar esos elementos comúnes y traer al frente unas estructuras maestras, unos patrones, que no sólo definan el comportamiento de estas historias, sino que además ayuden a construir nuevas. Éstas son las llamadas Tramas Maestras.

Una Trama Maestra presenta una fórmula para representar el viaje hacia la resolución del conflicto de la historia. Este conflicto se representa con una pregunta, llamada premisa dramática, y es la razón de la ascendente tensión durante la historia. El número de Tramas Maestras varía de autor a autor, siendo aquellas menores en cantidad más laxas para poder abarcar más historias.

En este proyecto se han tomado como base las *20 Tramas Maestras* de Ronald Tobias[8], que contiene una lista de puntos corta y precisa para lo que él categoriza como veinte tramas maestras. Él identifica: la Misión, la Aventura, la Persecución, el Rescate, el Escape, la Venganza, el Misterio, la Rivalidad, el Desvalido, la Tentación, la Metamorfosis, la Transformación, la

Maduración, el Amor, el Amor Prohibido, el Sacrificio, el Descubrimiento, el Precio del Exceso, la Ascensión y el Descenso.

3.3.1. Tramas de Acción

Se califican como tramas de acción aquellas donde la historia es movida por los eventos y no por la aspiración o los conflictos internos de los personajes. Se centran en la acción y no en la evolución de la persona.

En esta categoría se encuentran: la Aventura, la Persecución, el Rescate, la Huida, la Rivalidad, el Desvalido, la Venganza y el Enigma.

Cabe destacar que aunque no se busque la evolución del personaje , ello no implica que ésta no pueda darse, sino que no es el objetivo final de estas tramas.

3.3.2. Tramas de Personaje

Al contrario de las tramas de acción, las tramas de personaje siempre están movidas por la motivación de los protagonistas. Buscar a alguien, averiguar quién se es, una aflicción... Estas historias se apoyan en los conflictos internos para avanzar y presentar obstáculos.

Entre las tramas de personaje, están: la Misión, la Metamorfosis, la Transformación, la Maduración, el Ascenso, la Caída, la Tentación, el Precio del Exceso, el Sacrificio, el Amor y el Amor Prohibido.

4

Diseño de StoryDB

StoryDB es la base de datos que cumple con uno de los objetivos principales de este proyecto; proporcionar una estructura capaz de contener historias de forma coherente y concisa, que permita su uso a desarrolladores e investigadores por igual.

En este capítulo se explora el diseño e implementación de StoryDB. Ambos, diseño e implementación, se han desarrollado teniendo en cuenta los artículos mencionados en el capítulo segundo y la teoría expuesta en el capítulo tercero, en la cuál se ha basado ampliamente.

4.1. Estructura en Tres Actos

Como se ha explicado en el capítulo tercero, la estructura de la mayoría de historias puede ser dividida en tres actos, una forma de observar en un cuadro global los meta-datos que se nos presentan durante el transcurso de la historia.

Sin embargo, una historia es un texto. El texto de la historia se divide en capítulos, cada capítulo siendo una serie de párrafos que, a su vez, han sido construidos con oraciones.

En este apartado se explicará la construcción de StoryDB en base a estos dos conceptos; texto y estructura, empezando por sus unidades básicas hasta llegar a la composición total de una historia.

4.1.1. La unidad: Acciones

La unidad de un texto es la oración. A la hora de abordar una historia desde un punto de vista estructural, se podría decir que la unidad es una acción; *lo que hace* un elemento (o más) en un momento determinado.

Para guardar una historia en toda su extensión y poder analizarla después, se necesita almacenar cada acción que acontezca en ella.

Podemos determinar una acción por un verbo y los elementos, sean personajes, objetos o lugares, que toman parte en ella. Siguiendo esta lógica, se podría igualar una acción con una oración: ambas contienen verbo y una serie de elementos (sujetos, predicados y otras unidades léxicas, en el caso de las oraciones) que, en su conjunto, describen una acción.

Sin embargo, un ordenador es incapaz de procesar una oración como lo hace un humano sin las herramientas linguísticas que un humano ha adquirido durante su crecimiento. En el lenguaje humano, cada elemento del sujeto y del predicado representa un rol en la acción que se está describiendo. El humano detecta esos roles, les otorga un significado y de esa forma *comprende* el contexto, la coherencia y la función de todos los implicados en la oración. A esta comprensión de los roles se le llama semántica, y es lo que se necesita guardar para que un computador pueda procesar una acción.

Dado que no es objetivo de este proyecto el procesamiento de texto, se ha utilizado una herramienta llamada GluNet, explicada en el siguiente sub-apartado, para solventar el problema de la semántica.

Los roles semánticos que un elemento puede cubrir en una oración son: agente, co-agente, paciente, co-paciente, tema, co-tema, localización inicial, localización, destino, estímulo, meta, atributo, tópico, fuente, experimentador, recipiente, instrumento, herramienta, material, producto, beneficiario, extensión, predicado, tiempo, valor, trayectoria y pivote. Se puede aprender más de cada rol semántico en la documentación de VerbNet[9].

En la tabla *Action* de StoryDB se guardará el ID de la acción, el ID de la historia a la que pertenece y el verbo que constituye la acción. Después, por cada uno de los roles semánticos se guardará el objeto que corresponda al rol, si es que lo hay. Una oración nunca cubre todos los roles semánticos, a no ser que sea compuesta, en cuyo caso podría llegar a hacerlo, por lo que sólo aquellos roles que aparezcan quedarían cubiertos.

4.1.2. GluNet

GluNet[7] es una poderosa herramienta creada por TU Delft que unifica WordNet, VerbNet, FrameNet y ConceptNet, entre otros recursos, para crear una base de datos léxica y de conocimiento de «sentido común» de la lengua inglesa.

La unión de todas estas *Nets* provee a GluNet con la información sintáctica y semántica de innumerables verbos, característica que resulta crítica para este proyecto, ya que no se pretende analizar el texto de las historias.

La forma que tiene GluNet de almacenar la sintaxis y la semántica es mediante *Frames*, marcos, que engloban a verbos y sustantivos. Los verbos están guardados bajo números de identificación cruzada (un ID puede contener varias instanciaciones del mismo verbo, y varios verbos iguales pueden tener diferentes ID en la tabla principal), que nos informan de los diferentes significados que puede tener un verbo. Por ejemplo, *play* puede ser ambos jugar o tocar un instrumento.

Cada instanciación del verbo tiene unos marcos asignados, los cuáles guardan la estructura sintáctica que debe tener una oración con dicho verbo. Una estructura sintáctica sencilla podría ser NP V NP NP, donde NP es una preposición nominal (es decir, un sustantivo) y V es el verbo. Además, guarda una *string* para expresar (en su propio lenguaje) qué roles semánticos guarda cada elemento de la estructura y qué rol temático deben cubrir las palabras en esas posiciones.

Un rol temático es una clasificación sobre la palabra que indica sus características observables. Los roles temáticos son: abstracto, animal, animado, parte del cuerpo, comestible, comunicación, concreto, fuerza, prenda, humano, elemento de control, localización, máquina, organización, refl, región, escalar, sólido, sonido, substancia, tiempo y vehículo.

Por ejemplo, la palabra *caballo* tendría los roles temáticos animal, animado, comestible, concreto, sólido y vehículo, y podría aparecer en una oración cuyo verbo restrinja un rol semántico con alguno de estos roles temáticos, o cuyas restricciones temáticas estén vacías.

Enteder esta estructura y forma de manejar la lengua es uno de los mayores retos a la hora de trabajar con GluNet, pero resulta muy útil para el manejo de significados y oraciones. Para ejemplificar esto, observemos cómo GluNet guarda la información relativa al verbo *bring* (traer).

Primero, miraremos a los marcos del verbo, representados en la tabla 4.1.

FRAMES	NP	VERB	PREP	NP	NP	NP
158	Agent			Theme		
159	Agent		*against*	Theme	Destination	
160	Agent		*against*	Destination	Theme	
161	Agent			Theme	Initial Location	
162	Agent		*to*	Theme	Initial Location	Destination
163	Instrument			Theme	Destination	

Tabla 4.1 Marcos del verbo *bring* en GluNet.

La primera columna representa el número identificativo de cada marco para poder distinguirlo del resto en GluNet. El resto de columnas tienen como cabecera un elemento sintáctico, representando en su conjunto los elementos de los que se compone una oración. Sólo aquellas casillas coloreadas pertenecen al marco, aunque estén vacías de contenido, como en el caso de los verbos. Por ejemplo; una oración con el marco 161 del verbo *bring* tendrá la forma NP V NP NP.

Cada celda representa el rol semántico que se debe cubrir en cada elemento de la oración. Como ya se ha dicho, un rol semántico nos indica la importancia que tiene la entidad tomando parte en esa acción, de forma relevante a su posición para con el verbo dentro de la oración. Esto es, en esencia, lo que nos permite controlar si una oración será coherente o no.

Las preposiciones marcadas son fijas, por lo que las oraciones dependerán en su totalidad de las palabras elegidas bajo las restricciones semánticas expuestas.

El verbo *bring*, además, guarda las restricciones para los roles temáticos que deben cubrir los roles semánticos expuestos en los marcos. Es decir, qué categoría de palabras pueden cubrir cada rol semántico de este verbo. Las categorías son muchas, representando el conocimiento intrínseco que se tiene de lo que la palabra quiere representar. Por ejemplo, un caballo sería *animal*, pero también *transporte*, *concreto*, o incluso *comestible*.

Un elemento Agente (Agent) sólo puede ser de tipo *int_control*, una categoría que GluNet utiliza para referirse a fuerzas de control sobre objetos.

Los elementos Theme (Tema) hacen referencia al tema de la acción. Es decir, el objeto sobre el que se realiza o fuerza la acción. En otras palabras; el objeto directo, que sólo puede ser cubierto por *concretos*, en este marco.

Los elementos Localización Inicial (Initial Location) indican dónde empieza una acción. Sólo pueden ser de tipo *localización*.

Los elementos Destino (Destination) especifican dónde acaba la misma acción, y en este caso pueden ser cubiertos por palabras con las categorías *animado*, *localización* y *región*.

Por último, la categoría de elementos semánticos Instrumento (Instrument) no tiene ninguna restricción de rol temático.

4.1.3. Secuencias de Acciones

Ya se ha equiparado una oración con una acción. En la estructura de un texto, un párrafo es un conjunto de oraciones que expresa una misma idea, sea su definición o su desarrollo. Un párrafo contendrá al menos una oración y su extensión es indeterminada, según se necesite para ilustrar

la idea que se quiera representar. De la misma forma, una historia puede contener cualquier número de párrafos, los cuáles van enclaustrados en el esquema que se ha visto en la teoría.

Ya hemos equiparado una oración con una acción. De la misma forma, un párrafo será una secuencia (de acciones). Si bien un párrafo representa una idea, las secuencias representarán momentos o eventos de la historia. De nuevo, un computador no puede comprender una idea sin el conocimiento común y cultural del humano que lo lee. Sin embargo, teniendo ConceptNet en GluNet, y agrupando las acciones de esta forma, se le puede indicar el *concepto* del evento, la idea. Los conceptos no sólo servirán para ayudar al análisis de ideas, sino también para construir y delimitar las acciones en Puntos de Historia.

Para representar una historia necesitamos tres actos, equivalentes a una introducción, un nudo y un desenlace. Cada acto está compuesto por unos Puntos de Historia y el cuerpo del acto en sí mismo. Es decir; aquellos eventos comprendidos entre Puntos de Historia.

StoryDB tendrá una tabla por cada acto. Cada tabla acto estará definida por los Puntos de Historia y el cuerpo del acto que le corresponden. Todos los eventos de una historia serán guardados como secuencias de acciones, cuyo número identificativo se guardará en la columna del Punto de Historia en el que sucedan.

De esta forma, podemos unificar texto y estructura, manteniendo la cronología de los eventos de la historia sin perder de vista la posición que ocupan en el espectro global.

4.2. Tramas Maestras

Una Trama Maestra es un esquema que explica qué debe ocurrir durante el transcurso de la historia. De cara a la implementación, podríamos verlo como una meta-tabla que nos indica qué datos debemos guardar, qué clase de acciones deben tomar lugar y el espacio de historia que se debe cubrir.

Una Trama Maestra se representa desgranando una historia en Puntos de Historia.

4.2.1. Puntos de Historia (*Story Points*)

Un Punto de Historia (*Story Point*) es cada punto crítico en la estructura de tres actos. Estos serían: Status Quo, Detonante, Primer Punto de Giro, Punto Medio (opcional), Segundo Punto de Giro, Climax y Final.

Un Punto de Historia es crítico porque cambia completamente el ritmo y el rumbo de lo que se quiere contar. En ellos se dan los picos de tensión y los cambios de acto, mientras que el resto de eventos ayudan a la construcción.

Como se decía en el apartado anterior, una secuencia de acciones representará un evento de la historia. Sin embargo, un Punto de Historia es algo más importante. Es un momento decisivo. Al construir una historia siguiendo las Tramas Maestras, vemos que algunos Puntos de Historia van asociados con un concepto. Aprovechando GluNet y ConceptNet, podemos analizar las acciones asociadas a dicho Punto y determinar si cumple o no la estructura objetivo.

Echémosle un vistazo a la Trama Maestra de Búsqueda o Misión de Ronald Tobias:

- Núcleo de la historia: La evolución del héroe, su maduración.
- Meta del protagonista: Encontrar algo; una persona, un lugar, un objeto.
- Restricciones:
  - Acto I:
    - Detonante: Debe haber un incidente motivante. ¿Qué empuja al héroe a la misión?
  - Acto III:
    - Climax: Revelación. Lo que se descubre es (normalmente) distinto de lo que se buscaba.
    - Final: La misión puede completarse o no. El héroe ha cambiado.

A partir de esto se pueden inferir un conjunto de reglas que se aplicarán a cada Trama de Búsqueda o Misión:

Al terminar la historia, el protagonista debe haber cambiado, se haya completado o no la misión. Por tanto, se contrapondrán el protagonista del Status Quo con el del Final y se comprobará que haya habido, al menos, un cambio en su personalidad o creencias.

El Detonante debe ser un evento motivante, algo que establezca la meta para el personaje protagonista. En esta trama, el concepto de dicha meta es «Encontrar algo; una persona, un lugar, un objeto». Por tanto, el concepto del verbo contenido en la acción que representa a esta meta deberá coincidir sinonímicamente con *buscar.*

Estos conjuntos de reglas permitirán crear historias coherentes desde la herramienta de *input* de StoryDB, TaleBox. Guardarlas en StoryDB como información descriptiva de la historia aportará los datos necesarios para que el software pueda analizar la historia y su coherencia.

4.3. Espacio de la historia

Definimos como *Espacio* el conjunto de personajes, objetos y lugares que aparezcan en la historia, así como sus creencias, metas y transfondos. Es el espacio sobre el que se mueve la obra, establecida en el Status Quo, y que la acompañará hasta su final.

Para representar el espacio de la historia en StoryDB se han creado tres tablas: objetos, metas y creencias. Estas tablas recogen cada instancia de cada elemento en el espacio durante el transcurso de la historia; cada instanciación y cambio.

Denominaremos *Instancia* al estado de un elemento. La Instancia es como una fotografía del objeto en cada punto de la historia donde ha cambiado. Si extrajeramos todas las filas con un mismo ID, y las ordenáramos por su número de Instancia, podríamos observar la progresión de estados del elemento con ese ID durante toda la historia.

Lo que tienen en común estas tres tablas es su manejo.

En esencia, cada fila de estas tablas representa un elemento de la historia en un momento determinado de la historia. Un elemento se crea al aparecer en la historia por primera vez, obteniendo un ID único para la historia y el número 1 en su columna Instancia. Las instancias son marcadas de 1 a N por cada ID. La acción que lo registra deja su ID impreso, el ID de la secuencia al que pertenece, el cambio que ha realizado y en qué campo se ha dado el cambio. En caso de aparecer por primera vez, el cambio sería *Nuevo* y el campo se dejaría en blanco.

4.3.1. Personajes, objetos y lugares

Dadas las similares características que pueden tener los personajes, los objetos y los lugares, se han igualado estos elementos bajo una única tabla: Objetos.

Esta tabla contiene cada instancia de cada objeto durante el transcurso de una historia: cuándo se ha creado, sus cambios, etc. Esto permite analizar qué percibe el lector y el progreso global de los objetos durante periodos de la historia.

Los Objetos tienen Roles, pero dado que serán utilizados en las acciones, se ha diferenciado entre *Rol de Historia* y *Rol Temático*. El Rol de Historia puede ser Protagonista, Secundario, Antagonista, Objeto, Lugar o cualquier otra cosa que el autor vea necesario, mientras que el Rol Temático es la lista de roles temáticos que se atribuyen a este objeto, extraídos de GluNet, para poder utilizarlo correctamente en la tabla Acción.

Los tres tipos de objeto (personajes, objetos y lugares) pueden estar en una localización. Los personajes y los objetos se mueven durante la historia, apareciendo en diferentes lugares y yendo a otros. En el caso de los lugares, podemos decir que «El Valle Mágico está en la Región de Rondorr». Por tanto, la localización se aplica a todos los tipos de objeto. Este campo guarda el ID a un objeto lugar. A veces, una historia no especificará dónde están y quedará en blanco.

Los personajes, objetos y lugares también pueden tener una edad, una forma física que cambie durante la historia (una persona convertida en sapo), un estado (dormido, muerto, destruido), unos atributos (bonito, feo), unas características (miedoso, valiente), un inventario y un número de identidades por el que sean conocidos por otros objetos (un personaje conocido como El Rey por unos, y Leopold por otros).

Los personajes, además, guardan una lista de Metas y otra de Creencias, las cuáles se explican más adelante, representadas como listas de IDs. Los lugares y los objetos tendrán estos campos vacíos.

4.3.2. Metas

Las Metas son esenciales la construcción del SIG y las Tramas Maestras. Es lo que mueve a los personajes y da sentido a la historia. Para aportar todos los datos necesarios al software que utilice las historias de StoryDB, y permitir la construcción de un SIG automático, se han observado las siguientes características en una meta:

Tiene un estado. Puede ser *Posible*, *Imposible* o estar *Detenida*.

Tiene un rol en la historia. Algunas metas son roles de trama, otras son secundarias.

Tiene una serie de pasos. Ésta es una lista de IDs de la tabla de Acciones, por lo que al guardar una historia en StoryDB, se deberán especificar los pasos para lograr una Meta como acciones. Se ha interpretado como una pila, donde el último paso es el objetivo de la meta.

Por último, al igual que los objetos, una Meta puede tener varias identidades por las que se la conozca.

En algún punto de la historia, ciertos pasos serán cumplidos o modificados. Las modificaciones se pueden comprobar comparando distintas instanciaciones de la misma meta, pero los pasos realizados se guardarán como una lista aparte de IDs.

4.3.3. Creencias

Las Creencias ayudan a justificar tanto Metas como eventos durante la obra, definiendo no sólo su tránsito, sino también la personalidad de algunos personajes. Por ejemplo, en el cuento de Caperucita Roja, Caperucita piensa (cree) que las manos de su Falsa Abuela son muy grandes, sus orejas muy puntiagudas, sus dientes muy afilados... Esto nos da una visión general de la percepción del personaje y puede ser un punto de investigación interesante.

Para representar una Creencia, StoryDB guarda el ID del objeto al que se está referenciando, el nombre del atributo que se está cambiando y el nuevo valor que se le da.

Por ejemplo, si quisiéramos decir que Alicia (ID: 1) piensa que Bob (ID: 2) tiene treinta años de edad, esa Creencia se guardaría de la siguiente forma:

TABLA OBJETOS

- [1]
  - […]
  - Creencias: 1
  - Identidades: Alicia
  - […]
- [2]
  - […]
  - Edad: 25
  - Identidades: Bob
  - [...]

TABLA CREENCIAS

- [1]
  - Objeto ID: 2
  - Atributo: Edad
  - Es: 30

Esto permite guardar información contradictoria sobre los personajes sin manchar aquella que es la real. Se podría no tener el dato de la edad de Bob, pero al ser creencia de Alicia, se sabría que es una estimación aproximada y no real.

4.4. Implementación

Por último, se explica cómo se han integrado todos estos elementos para convertir la estructura objetivo en una base de datos funcional. El nombre de dicha base de datos es StoryDB.

4.4.1. Estructura general

StoryDB guarda las historias en un esquema árbol.

La tabla *Descripción* es el nodo raíz, donde se establecen el título, el/la/los autores, el tema, el orden en el que deben ser leídos los actos y, lo más importante, el ID de la historia en la base de datos. Este ID conecta el resto de tablas para formar la estructura completa, facilitando también las búsquedas.

El siguiente nivel es los Actos. Cada Acto de la estructura se representa con una tabla, cuyas columnas representan los Puntos de Historia del acto que representan y el cuerpo de la historia entre dichos puntos. Cada columna de una tabla de Acto guarda el ID de la Secuencia de Acciones que se le atribuye.

Cada acción guarda el ID de la historia y el ID de la secuencia a los que pertenece.

Por tanto, para buscar todas las acciones del Primer Punto de Giro de una historia cualquiera, cogeríamos su ID y miraríamos su tabla Acto_I, de la cuál obtendríamos el número de la secuencia guardado en la columna Primer Punto de Giro. Después, obtendríamos todas las acciones con este ID de secuencia.

La razón de que cada secuencia tenga un ID diferente, frente a tener un número para cada Punto de Historia y ahorrar tres tablas, es el poder hacer una búsqueda para todas las acciones de una historia o poder acceder directamente a parte de ella con el ID de su secuencia. Se ha elegido eficencia frente a espacio, ya que se espera que StoryDB contenga muchos datos en el futuro.

4.4.2. Estructura del Espacio

StoryDB guarda el espacio de cada historia en las tablas de Objetos, Metas y Creencias, ya descritas en un apartado anterior, y se conectan a la historia a la que pertenecen guardando el ID de ésta.

Estas tablas se diferencian, además, en que guardan no sólo un elemento del espacio, sino sus varias instancias. Esto imposibilita tener una sola clave primaria, ya que un objeto tendrá muchas instancias de sí mismo y, por tanto, habrá muchas filas con el mismo Objeto ID. Sin embargo, podemos usar el par de columnas *Objeto ID* e *Instancia* para diferenciar unas filas de otras, ya que por cada *Objeto ID* habrá una serie de *Instancia* del 1 al entero máximo que SQLite permite. Por ejemplo, si tenemos quince instancias del ID 1 y diecisiete del ID 2, habrá quince filas con Objeto ID 1 y diecisiete con ID 2, pero sólo dos filas para cada Instancia del 1 al 15, y dos filas con Instancia 16 y 17. Puesto de otra forma: [1] [1..15], [2] [1..17]. Al juntar ambas columnas, podemos comprobar que cada número será siempre único, porque [1, 13] no es lo mismo que [2, 13].

Esto se aplica a las tres tablas.

4.4.3. Tablas de Relación

Para permitir un análisis más profundo de las historias, StoryDB cuenta con tres tablas extra para guardar las relaciones entre acciones, metas y creencias. Cada tabla de relación guarda el ID de la historia a la que pertenece, los IDs de los elementos implicados y el tipo de relación que las une. Las relaciones sólo pueden darse entre elementos del mismo tipo; esto es: sólo dos acciones, dos metas o dos creencias.

La relación entre acciones permite la formación de oraciones más complicadas (subjuntivas, yuxtapuestas...) que permiten analizar el tiempo y sucesión de los eventos de la historia.

La relación entre metas garantiza una construcción del SIG más sólida, ya que en este tipo de esquema puede darse el caso de que una meta sea suficiente pero no necesaria para ayudar a otra en curso.

La relación entre creencias ayuda a crear perfiles más definidos de los personajes, cosa que puede ser de utilidad de cara al desarrollo de software e investigaciones.

4.4.4. Esquema Final

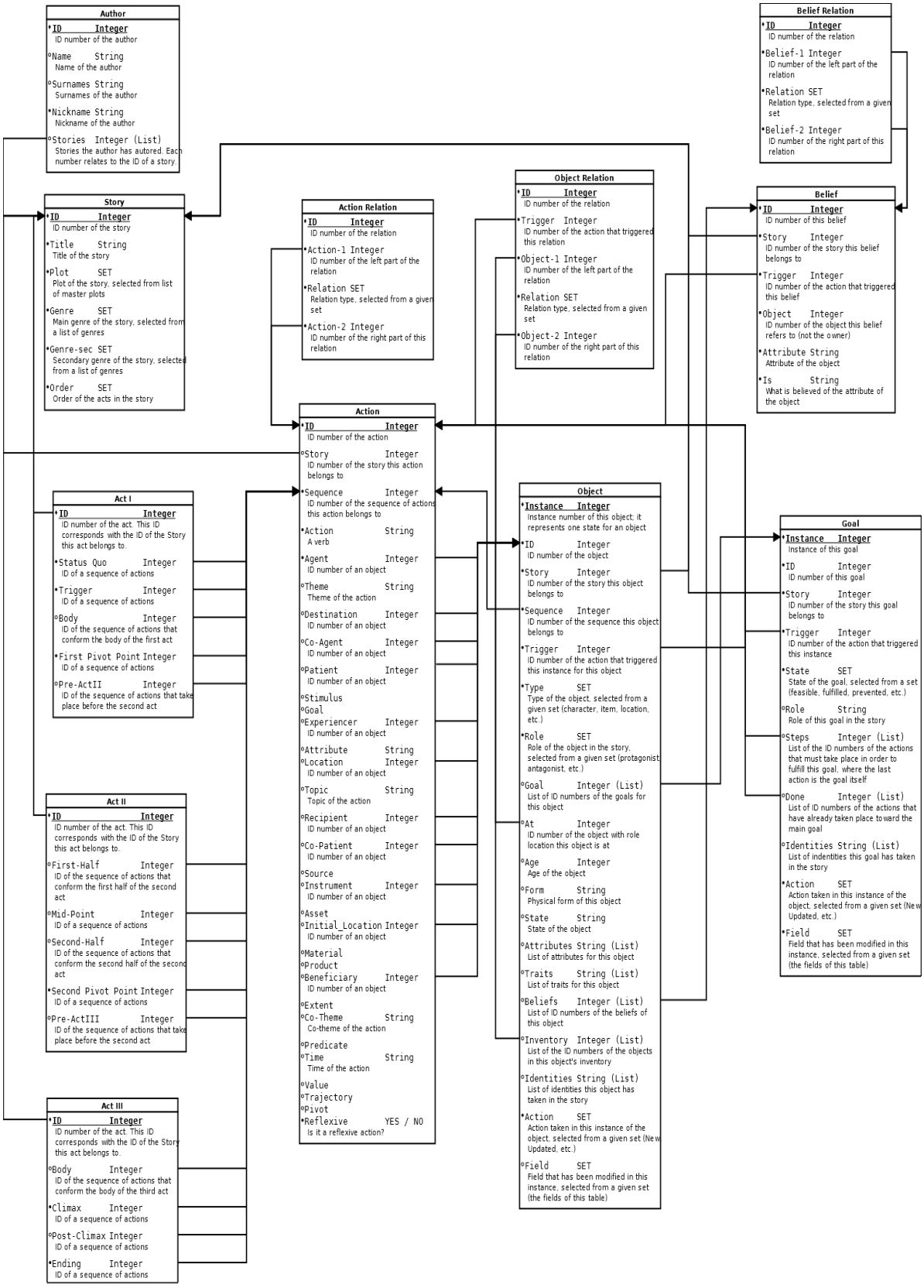


Figura 4.1 Esquema de la estructura de StoryDB (anexo C)

5

Diseño de TaleBox

TaleBox es una aplicación cuya funcionalidad principal es servir de *input* para StoryDB, permitiendo la creación de historias de acuerdo a los parámetros establecidos en el diseño de la estructura y servir de ejemplo para futuras aplicaciones.

Aunque la dirección principal de esta herramienta era crear contenido para StoryDB, se planteó la posibilidad de transformarla en un videojuego móvil que ayudara a promover la idea. Este capítulo explica el diseño de TaleBox, integrando ambos conceptos, herramienta y videojuego, como parte de su implementación.

5.1. Planteamiento de la funcionalidad

Con el diseño de StoryDB en mente, TaleBox debería permitir una creación secuencial de acciones en la historia que, al mismo tiempo, sean coherentes con la trama maestra y el género al que pertenece la historia. Es necesario, por tanto, desgranar el proceso de creación de una historia, desde una perspectiva *software*, para atender todas las necesidades de StoryDB sin dejar de hacer de TaleBox una herramienta flexible.

5.1.1. Acciones

Las acciones serán introducidas en TaleBox en secuencia. TaleBox proporcionará las palabras para garantizar la coherencia y consistencia de la historia, lo cuál es posible gracias a GluNet, que será parte íntegra del proyecto.

Ya que se quiere transformar TaleBox en un videojuego de cara al futuro, contaremos con las palabras en forma de cartas y varios usuarios para crear una historia, a los que llamaremos jugadores a partir de ahora.

Si los usuarios son jugadores, entonces las acciones serán *movimientos*. Se quiere que, por cada turno, un jugador pueda introducir más de un Movimiento en la historia.

Para crear un movimiento, un jugador utilizará cartas. Los Movimientos serán creados sintácticamente, a diferencia de las Acciones que funcionan de forma semántica. Esto quiere decir que el Jugador creará una oración con las cartas que se le han repartido. Estas cartas se entregarán de forma aleatoria por cada turno de jugador. Utilizando GluNet, TaleBox se asegurará de que al menos un Movimiento es posible en cada turno. Al formar un Movimiento correcto, TaleBox lo transformará en una Acción y lo guardará en la historia.

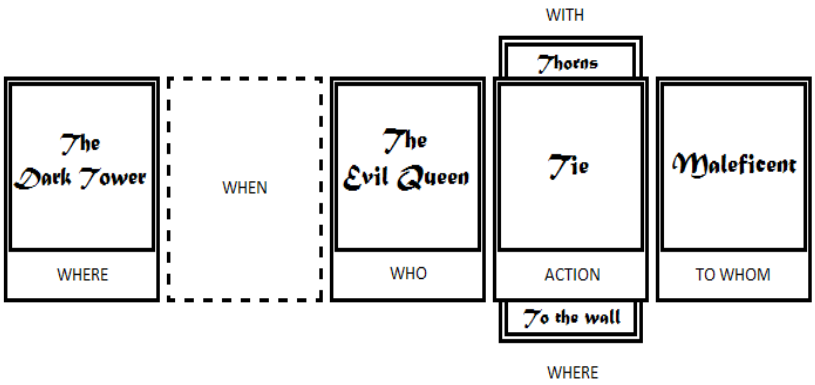


Figura 5.1 Boceto de un movimiento

5.1.1.1. Palabras / Cartas

Las cartas son extraídas de GluNet de acuerdo al/los género/s de la historia, ya que se trata de una herramienta de creación de historias y se espera que sean coherentes.

A la hora de crear un movimiento se tendrán en cuenta el dónde, el cuándo, los sujetos de la oración, el verbo, sus predicados y sus complementos. Se empezará rellenando el verbo, ya que es la unidad que condiciona toda la estructura de un movimiento. Con el verbo seleccionado, se habilitarán el resto de huecos necesarios para completar un movimiento correcto.

En muchos casos, el verbo cuenta con una preposición que, a su vez, marca uno o varios marco(s) concreto(s). Se le ofrecerá al jugador la posibilidad de elegir entre las preposiciones disponibles del verbo que ha elegido para facilitar encontrar el significado correcto del movimiento que quiera crear.

5.1.1.2. Usuarios / Jugadores

Los usuarios son necesarios para controlar quién introduce una acción en qué historia, y servirán como jugadores cuando TaleBox se convierta en un videojuego.

Los usuarios han sido introducidos en una copia de StoryDB para TaleBox. Guardan el ID de usuario, el pseudónimo y la contraseña de forma obligatoria. Se podrán introducir el nombre y apellidos reales de forma opcional. Estos datos quedarán atados a las historias creadas y acciones introducidas, de forma que se pueda atribuir la autoría a los jugadores de una historia.

Además, como dueño de una historia en juego, el usuario *administrador* podrá decidir si otros jugadores pueden unirse o no, cómo (por invitación, libre, etc.) y cómo serán aceptados en el juego; el administrador decide, a votación o automático.

Cuando un jugador haya perdido demasiados turnos, será automáticamente echado de la historia. Su participación quedará guardada en co-autoría, siendo la autoría para el creador original de la historia, aunque éste se marche. El autor será el administrador hasta entonces, que pasará al siguiente jugador que más tiempo lleve en la historia. En caso de empate, se mirarán los puntos.

5.1.1.3. Turnos

Cada turno, un jugador podrá poner uno o más movimientos en el juego. Esto equivale a poner una o más acciones en la historia. Cuando un turno es jugado, se avanza en la historia. La longitud de una historia se determinará al crearla y no podrá ser cambiada después.

Se denomina ronda a la secuencia de turnos donde todos los jugadores han jugado un turno. Tras un determinado número de rondas, se llegará a un Punto de Historia, un turno especial que será jugado por el jugador con más puntos acumulados. Este Turno especial será controlado por TaleBox, ofreciendo una serie de acciones especiales y más importantes, que encajen en la Trama Maestra de la historia.

Cada nueva ronda reordena los turnos de los jugadores de acuerdo a su puntuación.

Si un jugador entra en la historia durante una ronda, su turno se colocará al final de la siguiente, para asegurar que si un jugador sale y vuelve a entrar no tenga dos turnos, y para no hacer infinitas las rondas.

Las cartas se reparten para cada turno. Aquellas no utilizadas *no* se guardan para el siguiente turno. Esto evita problemas de coherencia, como que un personaje muera en la ronda pero su

carta siga en la mano de algún jugador. Todas las cartas se reparten de acuerdo al turno a punto de ser jugador, teniendo en cuenta el estado actual del juego.

Al terminar un turno, se abrirá una ventana de tiempo en la que el resto de jugadores pueden *retar* un movimiento de los jugados. Las razones para retar el movimiento son falta de coherencia o consistencia con la historia. Retar un movimiento hará entrar en juego la mecánica de puntuaciones, parte vital del juego. El jugador retado deberá justificar el movimiento con otro movimiento, que no podrá ser retado. El objetivo de los retos no sólo es lúdico, sino que busca incitar a los jugadores a crear más transfondo para sus personajes e historias, aumentando la calidad de las historias introducidas en StoryDB.

5.1.2. Mecánicas

Las dos mecánicas más importantes de este juego son la puntuación y la selección de cartas.

La puntuación se utilizará como elemento lúdico del videojuego, incitando a los jugadores a querer seguir jugando y dándoles una razón para esforzarse al máximo con sus historias, ya que aquél con más puntuación consigue los turnos especiales. Los puntos se entregan de la siguiente forma:

- 1 punto por turno jugado.
- 1 punto por reto completado.
- -1 punto por reto fallado o turno pasado.
- -2 puntos por turno perdido.

La selección de las cartas, por otra parte, dependerá de varios factores: el género, la trama maestra y el estado de la historia en el momento de ser seleccionadas.

Para facilitar la selección por género, se ha creado una rama de GluNet para TaleBox con una tabla extra que *mapea* cada palabra a un género de historia. Dado que dicho objetivo no está contemplado en los objetivos de este proyecto, debido a su gran envergadura (GluNet contiene más de dos millones de palabras), se han utilizado las cuatro mil más utilizadas en el lenguaje inglés y se han incluido en todos los géneros.

TaleBox debe ofrecer al menos una acción completa cada vez que mande las cartas al jugador. Para asegurarlo, se deben comprobar los roles temáticos y semánticos de cada palabra, un trabajo muy ineficiente de hacerlo sobre GluNet de forma directa. Para ayudar con este trabajo, se ha añadido otra tabla al GluNet de TaleBox: roles temáticos. Esta tabla contiene cada rol temático como una columna y marca aquellos que poseen cada palabra de la base de datos. Ha sido mapeado utilizando un script en Python.

Por último, se considerará el estado de todos los elementos del espacio de la historia. Se escogerán sólo aquellos cuyo uso sea coherente (personajes no muertos). Se dará prioridad a los elementos del espacio de la historia, aunque se garantizará la inclusión de elementos nuevos; cartas de sustantivos extraídas de GluNet.

5.1.3. Historias

La historia es el elemento más importante en este proyecto. Está compuesta por secuencias de acciones correlacionadas por las características de la propia historia. En el caso de historias jugadas, está compuesta por movimientos jugados por los jugadores. Por tanto, es muy importante delimitar la libertad que tendrán los usuarios de TaleBox al crear una historia sin cuartar su creatividad.

A la hora de empezar una historia con TaleBox, los usuarios podrán elegir:

- El título de la historia.
- La trama maestra, a elegir de la lista proporcionada en el apartado de teoría. Determina el esquema que seguirá la historia.
- El género o géneros de la historia, ya que una historia puede tener más de un género, determina el tipo de cartas que se seleccionarán en cada turno. Se elegirá a partir de la siguiente lista:
    - Acción, aventura, comedia, crimen, fantasía, historia, ficción histórica, horror, realismo mágico, misterio, paranoia, filosofía, política, romance, saga, sátira, ciencia ficción, thriller, urbano y western.
- La longitud de la historia, que determinará cuántas rondas tendrá cada acto de la historia. A elegir entre: relato, novela corta, novela y épico.

TaleBox expandido como videojuego, contará con las siguientes opciones adicionales:

- Número máximo de jugadores por historia. No se podrá cambiar durante el transcurso de la historia. Se permitirá un mínimo de uno (para historias de un solo autor) y un máximo de diez.
- Número de movimientos por turno. Es decir, cuántos movimientos puede jugar un jugador en su turno. De un mínimo de uno a un máximo de seis. Los jugadores sólo estarán obligados a jugar un movimiento, independientemente de cuántos se les permita, y no afectará a la puntuación. No se podrá cambiar durante el transcurso de la historia.
- Límite de tiempo por turno, para que el juego siga en marcha y echar a los jugadores no activos. No se podrá cambiar durante el transcurso de la historia.
- Privacidad de la historia: pública, restringida o privada. Las historias públicas podrán leerse en el panel principal del juego. Las restringidas sólo serán accesibles si se es amigo de un jugador en dicha historia. Las privadas sólo serán accesibles para los propios jugadores de la historia.
- Forma de unión: libre, por invitación o cerrada. Libre implica que cualquier jugador puede solicitar unirse a la historia. Por invitación, restringe a los jugadores invitar a otros a entrar. Una historia cerrada no permitirá la unión de nuevos jugadores, aunque haya espacio.

- **Política de aceptación**: decide el administrador, votación o automáticamente. En la votación, los jugadores actuales de la historia deciden si el nuevo jugador puede entrar. En caso de empate, el voto del administrador cuenta doble.

Compuesto por una secuencia de turnos y movimientos, la idea conceptual de la presentación de una historia es la siguiente:

Figura 5.2 Concepto de la representación de una historia

5.2. Cliente - Servidor

Dado que el objetivo de la aplicación es poder ser utilizada a distancia y entre varias personas, un enfoque cliente-servidor es necesario. A continuación se explica el enfoque tomado para estructurar y repartir la carga del diseño entre ambas partes.

5.2.1. Estructura

Debido a la clara diferenciación entre datos y lógica en el diseño de TaleBox, se decidió utilizar una arquitectura a tres capas.

La capa de datos se encargará del manejo de ambas bases de datos, StoryDB y GluNet, en sus versiones para TaleBox.

La lógica se ha dividido en módulos: lógica del usuario, lógica de la historia, lógica del juego, lógica del server y croupier. Estos cinco módulos, repartidos y replicados entre el cliente y el servidor, han sido delimitados para funciones específicas, de forma que actualizar la aplicación o tener varias versiones sea más sencillo y mantenible a largo plazo. Esto también permitiría utilizar TaleBox con ambas funciones; videojuego y herramienta de introducción de historias, ya que el objetivo es delimitar las funciones para que su uso sea genérico y reutilizable.

Por último, la capa de interacción con el usuario se ha reservado para el cliente.

5.2.2. Cliente

Ya que el cliente se va a instalar en dispositivos móviles, es imperativo que éste ocupe el menor espaco y requiera los menos recursos posibles. Por este motivo, se ha decidido que el cliente tenga la menor carga de la aplicación. Su función será la de recibir los datos del usuario, hacer unas comprobaciones mínimas y mandar los datos al servidor, que se encargará de procesar el resto.

El cliente, por tanto, contará con los módulos de historia, usuario y juego, que se ocupan de preparar los datos y hacer unas últimas comprobaciones.

Su capa de datos será la conexión con el servidor, llamada *ClientGate* o *Puerta del Cliente* para esta aplicación.

5.2.3. Servidor

El servidor será la parte de TaleBox que más carga reciba. Contendrá todos los módulos de lógica y se ocupará de todas las operaciones necesarias para llevar a cabo la funcionalidad de la aplicación. Además, preparará los datos tanto para la capa de datos como para el cliente.

Su capa de interacción con el usuario será, en este caso, el módulo de conexión con el cliente, *ServerGate* o *Puerta del Servidor.*

La comunicación servidor-cliente será a través de strings, separadas por tokens, para agilizar las transferencias. Estos token serán acordados al inicio de la conexión para asegurar que se puedan cambiar sin necesidad de actualizar el cliente.

6

Implementación de TaleBox

TaleBox es una aplicación cuya funcionalidad principal es servir de *input* para StoryDB. Su implementación, sin embargo, se ha enfocado en el desarrollo de un videojuego multijugador de cartas por turnos para dispositivos móviles, de forma que se permita la creación colaborativa y creativa de historias.

En este capítulo se detalla la implementación de TaleBox como un videojuego.

6.1. Software

Se ha elegido la plataforma de desarrollo Unity para desarrollar TaleBox. Unity es una herramienta muy flexible que facilita la creación de videojuegos, tanto 2D y 3D, y su distribución a distintas plataformas de forma gratuita. En este caso, el objetivo de TaleBox es ser una aplicación móvil para Android.

Se ha utilizado C# para el *scripting* en Unity.

Para mantener la coherencia, ya que cliente y servidor comparten clases objeto en gran medida, el servidor de la aplicación ha sido programado en Visual Studio utilizando C#.

Por último, las versiones de StoryDB y GluNet para TaleBox han sido implementadas en SQLite, como sus versiones originales.

La razón de utilizar SQLite es que es libre para cualquier propósito. No necesita un proceso de servidor para funcionar, permite manejar ficheros de hasta 2 TeraBytes, su tamaño es pequeño y no necesita configuración (lo cuál ayuda en la portabilidad del server del juego), no fuerza restricciones en los tipos de datos y, ya que no hay negociación cliente-servidor, es capaz de acceder a la base de datos dos o tres veces más rápido que una base de datos MySQL.

Por otro lado, también hay que tener en cuenta que no todas las *queries* SQL o su sintáxis están soportadas por SQLite, necesita mucha memoria para ejecutarse (256 bytes de RAM por cada MegaByte, aproximádamente) y tiene bloqueo lectura/escritura, lo cuál retrasará escribir en ella.

Estas limitaciones, sin embargo, pueden ser subsanadas adaptando las *queries* a la sintáxis de SQLite y controlando las lecturas y escrituras. Respecto al uso de memoria, una historia no debería ocupar más de unos cuantos KyloBytes cada una. Si se diera el caso de que la base de datos creciera hasta un punto insostenible por el servidor, los datos se migrarían a otro soporte de datos.

6.2. Servidor

El servidor de TaleBox es la parte de la aplicación que más cargo de trabajo va a tener, ya que se quiere garantizar que las necesidades del cliente puedan ser soportadas por el mayor número de dispositivos móviles posible.

Veamos en detalle qué hace cada parte del servidor.

6.2.1. Capa de datos

La capa de datos del servidor ha sido adaptada para servir a ambos GluNet[7][D] y StoryDB[E]. La documentación de ambos se puede encontrar en los anexos de esta memoria.

La capa de datos se ha desarrollado en base a una clase raíz, DB_Main, de la que heredarán las clases encargadas de manejar las bases de datos. Las clases se han dividido de acuerdo a la base de datos a la que sirven.

6.2.1.1. DB_Main

DB_Main proporciona seis funciones auxiliares que deben estar presentes en todas las clases para el control de la capa de datos de TaleBox.

`openDB(string DBname)` instancia la conexión con la base de datos antes de empezar una *query*. Su único parámetro es el nombre de la base de datos que se quiere abrir, el cuál será proporcionado por las clases que hereden de DB_Main.

Durante una función de manejo de datos, puede darse el caso de que se necesite utilizar varias *queries* en escritura o lectura, o una combinación de ambas. Se observó que, para manejar los Objetos `IDataReader` de lectura, eran necesarios una serie de pasos cada vez que se quisiera abrir un nuevo canal de lectura que podrían ser evitados de tener una función general que se ocupara de ello. Para esto fueron creadas las dos funciones siguientes:

```
protected IDataReader Query(string query) {

              IDbCommand cmd = ((IDbConnection) connection).CreateCommand ();
              IDataReader aux;

              try {
                     cmd.CommandText = query;
                     aux = cmd.ExecuteReader ();
              } catch (Exception e) {
                     closeDB ();
                     Console.WriteLine(e.Message);
                     cmd.Dispose();
                     cmd = null;
                     return null;
              }

              cmd.Dispose();
              cmd = null;
              return aux;
}

protected bool Transact(string query) {
              bool ok = true;

              SQLiteCommand command = connection.CreateCommand ();
              SQLiteTransaction transaction;
              transaction = connection.BeginTransaction ();
              command.Connection = connection;
              command.Transaction = transaction;

              try {
                     command.CommandText = query;
                     command.ExecuteNonQuery();
                     transaction.Commit();
              } catch (Exception e) {
                     Console.WriteLine(e.Message);
```

```
                    ok = false;
                } finally {
                    command.Dispose ();
                    command = null;
                    transaction.Dispose ();
                    transaction = null;
                }
            return ok;
}
```

Programa 6.1 Algoritmos auxiliares para el manejo de *queries*

Ambas funciones son parte de DB_Main, que contiene la instanciación de los atributos `SQLiteConnection connection` e `IdataReader reader`. El hecho de que estas funciones se ocupen de abrir y anular los punteros de conexión, lo cuál puede ser una gran fuente de *bugs*, permite eliminar una cantidad considerable de líneas en la implementación del manejo de los datos.

Por último, DB_Main contiene otras dos funciones `parseNullInt(int index, int defaultValue)` y `parseNullString(int index, string defaultValue)`, cuya función es ayudar en la obtención de datos de celdas cuyo valor es `null`, lo cuál es bastante frecuente en GluNet.

6.2.1.2. GluNet

GluNet_Main es la clase que se ocupa del manejo de datos de GluNet. En este caso, ya que la función de la base de datos es proporcionar palabras para la construcción de historias, GluNet_Main sólo lee y extrae palabras, de acuerdo a unos parámetros.

Sus dos funciones más importantes son `public Dictionary<int, List<Verb>> getCoreVerbs()` y `public List<Card> getRandomCards(int n, List<string> genres, string type, List<string> thematic_restriction)`, las cuáles se ocupan de obtener todos los verbos utilizables por TaleBox (aquellos más importantes del habla inglesa) independientemente del género de la historia, y cartas aleatorias en base al tipo de palabra que se busca, el género de la historia y las restricciones temáticas, respectivamente.

6.2.1.3. StoryDB

El manejo de StoryDB es más complejo. A la hora de tratar StoryDB, tanto para lectura como para escritura, es necesario dividirla en cinco apartados: Acciones, Objetos, Historias, Turnos y Usuarios. Esto vuelve mucho más sencilla la modularización, facilitando futuros distintos usos para el servidor de TaleBox.

SDB_Actions se ocupa de la lectura y escritura de nuevas Acciones en la base de datos, así como de la conexión entre ellas para oraciones más complejas.

SDB_Objects se ocupa de leer el espacio de la historia; desde introducir nuevas instancias de un elemento determinado hasta obtener una instancia de un objeto en un momento determinado de la historia.

SDB_Stories contiene las funciones para la creación de historias, la obtención de aquellas ya creadas y el manejo de los ID que identifican las secuencias de acciones de cada acto.

SDB_Turns es la clase que administra los datos de los juegos en marcha, los turnos y las puntuaciones. Es el módulo de la capa de datos que posibilita la funcionalidad de videojuego de TaleBox.

SDB_Users se encarga de administrar los datos de los usuarios, su creación y la confirmación de credenciales para entrar al sistema.

Con la adición de los turnos y usuarios a TaleBox, la estructura de StoryDB también ha sido modificada para permitir la inclusión de todos estos datos. La forma final de StoryDB es la siguiente, y puede encontrarse una imagen amplificada en los anexos de esta memoria:

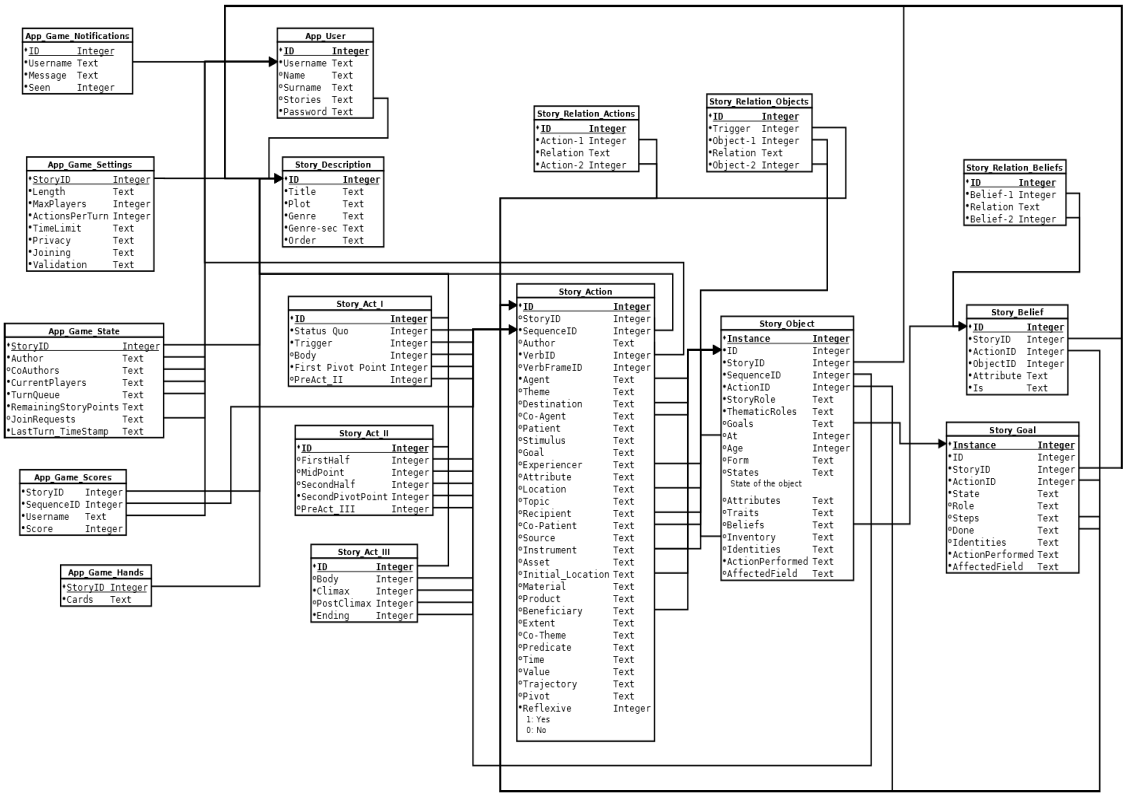


Figura 6.1 Estructura final de StoryDB

6.2.2. Lógica

Los módulos de lógica de TaleBox son los encargados de asegurar que tanto el servidor como el juego funcionen perfectamente. Comunican al cliente con la capa de datos y se encargan de administrarla, de forma que todo lo guardado sea coherente con las especificaciones que se han dado en el capítulo anterior.

Para esto se han dividido las funciones en cinco módulos más la conexión con el cliente, que también sirve como *main* del servidor. Estos son: Croupier, GameLogic, ServerLogic, StoryLogic, UserLogic y ServerGate.

6.2.2.1. Croupier

El módulo de lógica Croupier es, sin duda, el más importante respecto a TaleBox como videojuego. Se ocupa de la inteligencia detrás del reparto de cartas por turnos, lo cuál implica hacer un repaso del estado actual de la historia y encargarse de que todos los roles semánticos y temáticos de al menos un marco para un verbo en una mano de cartas sean satisfechos, de forma que los jugadores siempre tengan al menos un movimiento posible.

Croupier tiene como atributos un Diccionario con la lista de verbos núcleo del habla inglesa, extraída de GluNet al iniciarse el server para ahorrar tiempo de ejecución posterior, y la lista de llaves identificativas de ésta para acelerar su procesamiento. Se ha considerado que todos estos verbos deberían estar disponibles para los jugadores, independientemente del género en el que estén jugando, y es la piscina de la que se eligen los verbos actualmente. Los verbos se guardan de una forma muy específica, de acuerdo a la clase *Verb*, que a su vez guarda todos los marcos en objetos *VerbFrame*. Cada verbo es guardado en este Diccionario bajo la llave identificativa de su ID en GluNet.

Croupier es llamado en cada turno para barajear una nueva mano de cartas de una historia determinada. La función `public static string dealCards(int storyID)` se ocupa de esto.

Debido a su extensión, el código se incluye en la página siguiente, seguido de una explicación del algoritmo.

```
public static string dealCards(int storyID) {

        SDB_Stories storiesDB = new SDB_Stories();
        Story story = storiesDB.getStory(storyID);
        string storyPoint = story.getCurrentStoryPoint();
        List<string> genres = story.getGenres();
        int actionsPerTurn = story.getActionsPerTurn();

        int n_Verbs = actionsPerTurn + actionsPerTurn / 3;
        int n_genericCards = actionsPerTurn * 3;
        int n_storyCards = actionsPerTurn * 5;

        List<Verb> verbs = new List<Verb>();
        List<Card> storyCards = new List<Card>();
        List<Card> genericCards = new List<Card>();

        if (n_Verbs > 0)
        {
                // Get n random verbs
                verbs = getRandomVerbs(n_Verbs, storyPoint);

                List<string> thematic_restrictions = new List<string>();
                int adjCount = 0;
                int advCount = 0;

                 // Group all unit type - restrictions
                foreach (Verb v in verbs)
                        foreach (VerbFrame f in v.getFrames()) {
                                // Get rid of VERB, PREP and LEX restrictions.
                                VerbFrame auxFrame = f.isolate();
                                for (int i = 0; i < auxFrame.syntaxRoles.Count; i++){
                                        if (auxFrame.syntaxRoles[i] == "ADJ") {
                                                adjCount++;
                                        } else if (auxFrame.syntaxRoles[i] == "ADV") {
                                                advCount++;
                                        } else {
                                                List<string> restrictions =
                                        v.getRestrictionsFor(auxFrame.semanticRestrictions[i]);
                                                foreach (string r in restrictions)
                                                if (!thematic_restrictions.Contains(r))
                                                                thematic_restrictions.Add(r);
                                        }
                                }
                        }
                        storyCards = getStorySpaceCards(storyID, n_storyCards,
restrictions, total);
                        n_genericCards += n_storyCards - storyCards.Count;

                        genericCards = getGenericCards(n_genericCards - adjCount/2 -
advCount/2, genres, "NP", thematic_restrictions);
                        genericCards.AddRange(getGenericCards(adjCount/2, genres, "ADJ",
thematic_restrictions));
                        genericCards.AddRange(getGenericCards(advCount/2, genres, "ADV",
thematic_restrictions));
        }

        string output = "";


        output += Utils.attach(verbs, ServerGate.arg_separator);
        output += ServerGate.func_separator;
        output += Utils.attach(storyCards, ServerGate.arg_separator);
        output += ServerGate.func_separator;
        output += Utils.attach(genericCards, ServerGate.arg_separator);
        return output;


}
```

Programa 6.2 Algoritmo de generación de manos de cartas

Los pasos que sigue el algoritmo son los siguientes:

- Pide al módulo de datos SDB_Stories una copia completa de la historia que va a tratar con el ID que ha llegado del cliente.
- De la historia, obtiene el Punto de Historia en el que se encuentra, los géneros a los que pertenece y el número de movimientos por turno que tiene. Estos datos son vitales a la hora de repartir las cartas.
- Calcula el número de verbos, cartas genéricas (extraídas de GluNet) y cartas de historia (extraídas del espacio de la historia) que habrá en la mano. Para esto, utiliza el número de movimientos por turno.
    - Nº de verbos: movimientosPorTurno + movimientosPorTurno/3
    - Nº de cartas genéricas: movimientosPorTurno*3
    - Nº de cartas de historia: movimientosPorTurno*5
- Obtiene aleatoriamente el número de verbos que se ha calculado.
- Recoge las necesidades de cada verbo:
    - Aisla cada marco de cada verbo. Esto es: crea una copia del marco en la que se han eliminado las preposiciones y unidades léxicas (TaleBox trata ambas de la misma forma) y el elemento Verbo de la sintaxis, de forma que sólo queden los sustantivos; los únicos elementos con necesidades semánticas y sintácticas, los adjetivos y los adverbios, que son las cartas a repartir aparte de los verbos.
    - Comprueba los roles sintácticos que quedan para el marco.
        - En el caso de los adverbios y los adjetivos se limita a sumar uno en la cuenta de cuántos se necesitan de cada uno.
        - Para los sustantivos, obtiene del verbo las restricciones semánticas. De cada restricción semántica obtendrá sus restricciones temáticas. Estos roles temáticos serán añadidos a la lista `thematic_restrictions` si no se encontraban ya en ella.
    - Llama a la función `getStorySpaceCards(...)`, contenida en el propio módulo Croupier, para obtener las cartas del espacio de la historia. Su objetivo es revisar el estado de cada objeto y decidir si es apto para incluirse en la mano. Después, elegirá aleatoriamente de aquellos obtejos que hayan pasado la criba tantos como se le hayan indicado, o todos, de no haber suficientes.
        - Esta función está comentada en el servidor, ya que su implementación aún no se ha terminado.
    - Para asegurar que haya tantas cartas como eran necesarias, se añade al número de cartas genéricas la resta del número de cartas de historia que se necesitaban menos las que se han cogido de verdad.

- Se llama a la función `getGenericCards(...)` tres veces; una para sustantivos, otra para adjetivos y otra para adverbios. Esta función utiliza el módulo de datos GluNet_Main para coger palabras de GluNet que cubran las necesidades que se le han solicitado. Estas son: el tipo de palabra (sustantivo, adjetivo o adverbio), el género de historia al que deben pertenecer y los roles temáticos que se deben cubrir con ellas (sólo para sustantivos).
- Por último, las tres listas se *codifican* a su formato string y se devuelven como resultado de la función.

6.2.2.1. GameLogic

Las dos funciones más importantes del módulo de juego, aparte de extraer las acciones de una historia y traducirlas en movimientos para enviárselas al cliente, son introducir nuevas acciones y crear nuevos objetos para la historia.

La función `newActions(...)` se encarga de recibir los nuevos movimientos y traducirlos a acciones de historia. Para esto, vuelve a comprobar que el movimiento sea correcto utilizando la función `isValid()` de la clase *Move*, que comprueba que todas las cartas, en las posiciones en las que se encuentran, corresponden con los roles temáticos y semánticos del marco seleccionado para el verbo. Después, crea un diccionario cuyas claves son *string*s. Estas *string*s representarán cada rol semántico existente en GluNet. Se añaden al diccionario con *string*s vacías como datos, ya que no todos los roles semánticos serán cubiertos por la acción.

Aquí es donde entra en juego la función `parseObject(int storyID, int sequenceID, int actionID, Card card)`. Esta función toma los datos del estado actual de la historia y una carta, y devuelve su ID como objeto en la base de datos.

Comrpueba si la carta ya se encuentra en el espacio de la historia, en cuyo caso analiza la carta, crea una nueva instancia para el objeto con la información aportada por la acción y devuelve el ID. En caso contrario, analiza la carta y sus roles temáticos, la añade a la base de datos como un objeto o un personaje en la historia, y devuelve el ID que se acaba de crear para este objeto.

Esta función no sólo se ocupa de crear los nuevos objetos, sino que también los actualiza creando nuevas instancias para ellos. Para actualizarlos, le basta con llamar a `updateObject(...)`, en el módulo SDB_Objects, que recibiendo el nombre de la columna a actualizar y los datos a añadir, se ocupa de todo el proceso.

De haber pasado una carta nula, devuelve una string vacía. La razón de que devuelva s*trings* y no *integer*s es debido a que las *queries* se hacen con *string*s, y hacerlo de forma directa con este tipo de datos previene *bugs* innecesarios.

Una vez procesada la carta, `newActions(...)` guarda el ID que ha recibido en el campo semántico que se le había atribuido a esa carta y pasa a la siguiente. Hará lo mismo hasta haber procesado todas las cartas de ese movimiento, ahora transformado en acción.

Después, lo único que tiene que hacer es llamar a `addAction(...)` en SDB_Actions, que cogerá el nombre del autor, los datos de la historia, del verbo y el diccionario semántico y se encargará del resto.

Hará lo mismo para cada movimiento que le haya llegado. Si todo ha ido bien, el cliente recibirá un mensaje de confirmación por parte del servidor.

Tras esto, el módulo de juego llamará a la función encargada de generar el siguiente turno.

6.2.2.1. StoryLogic

El módulo de historias se ocupa de lo más básico de TaleBox: crear nuevas historias, sacar historias de la base de datos y manejar los turnos de juego.

Al registrar una nueva historia, tras ingresar los datos a StoryDB, genera un nuevo turno para el primer jugador, siempre el autor. Este turno genera una mano de cartas que se guarda en StoryDB y una notificación para el jugador en cuestión de que su turno ha empezado, tras lo cuál se inicia el temporizador del turno.

También contiene la función que genera el siguiente turno y la siguiente ronda; `nextTurn(...)` y `Queue<string> newRound()`, respectivamente.

`nextTurn(...)` se encargará de sumar o restar los puntos necesarios al jugador del cuál era el turno que ha acabado, dependiendo de si ha entregado sus movimientos, ha pasado el turno o lo ha perdido (en cuyo caso recibirá una notificación), y generará el del siguiente jugador en la cola. Todo esto quedará reflejado en la base de datos; el porcentaje de turnos perdidos/jugados del jugador, el punto de la historia y a quién pertenece el nuevo turno.

En caso de que no quedaran jugadores en la cola de juego, se llamaría a la segunda función. `Queue<string> newRound()` recoge las puntuaciones de todos los jugadores para la historia seleccionada hasta el momento, los ordena de mayor a menor e introduce la nueva cola en la base de datos. Después, genera el turno para el primero y se le envía la notificación.

6.2.2.1. UserLogic

El módulo de usuario es el módulo de lógica más sencillo de todos, pues sólo se ocupa de registrar usuarios, comprobar sus credenciales y mandarle al cliente las notificaciones para el user ID que se le haya solicitado.

6.2.2.1. ServerLogic y ServerGate

El módulo de lógica ServerLogic y la clase ServerGate están estrechamente relacionados, ya que ambos se ocupan del funcionamiento interno del servidor.

ServerGate es la clase que inicia el servidor como aplicación en la máquina, pero también se ocupa de escuchar al cliente. Es la puerta por la que pasa toda la información y comunicación entre el servidor y el cliente de TaleBox.

Al inciarse, ServerGate pide al módulo Croupier que extraiga los verbos núcleo de GluNet y los guarde en memoria. Tras esto, inicia la escucha al cliente en cinco *thread*s distintos. Cubriremos esto en el siguiente apartado.

También inicia un sexto *thread* que se ocupa del funcionamiento interno del servidor. Este *thread* duerme durante un minuto, el tiempo mínimo para un turno de TaleBox, y después

comprueba todos los turnos en progreso de la base de datos. Dicha función, `bool tickTheClocks()`, se encuentra en ServerLogic.

Extrae los tiempos de inicio y finalización de cada turno y comprueba que ninguno haya sido *perdido*; es decir, ninguno sobrepasa el tiempo límite que se fijó al crear la historia. De haberlo, lo añade a una lista de turnos perdidos, atributo principal de ServerLogic, y devuelve *true* al hilo principal del servidor.

Con esto, el servidor volverá a llamar a ServerLogic para que se ocupe de generar los nuevos turnos para dichas historias (como se explicó en el subapartado StoryLogic) y volverá a dormir.

6.2.3. Comunicación con el cliente

La comunicación con el cliente se hace bajo el protocolo TCP, ya que se necesita una conexión estable con el cliente mientras se procesan los datos en el servidor, y esto podría tomar varios segundos.

La comunicación se basa completamente en *string*s, intercambiadas entre servidor y cliente. Cuando el cliente requiere de una función del servidor, le manda el identificador de dicha función junto con los parámetros que requiera para trabajar.

Este identificador y los datos requeridos son *cifrados* mediante tokens, que separan el identificador de los diferentes parámetros, también en formato *string*. Estos tokens han sido acordados al abrir la conexión con el servidor, de forma que se puedan cambiar más adelante de ser necesario.

TaleBox y su servidor tienen muchas clases en común, y sus datos han de ser transferidos de uno a otro en este formato de *string*s. Para ello, se ha equipado todas las clases de TaleBox con tokens para que su información pueda ser transformada a texto. Dado que algunas clases contienen a otras (por ejemplo, *Move* contiene una lista de *Card*), los tokens han sido distribuidos de acuerdo a la jerarquería que forman. Actualmente, cambiar uno de estos tokens implicaría cambiar el cliente, pero se espera que en un futuro esto pueda ser comunicado ente ambas partes al iniciar la comunicación.

Cuando se necesita enviar datos de una parte a la otra, se le pide a las clases que se codifiquen. Dichas codificaciones se atan con los tokens de comunicación y se mandan. Al llegar al otro lado, las mismas clases se encargan de decodificar la información como parte de su instanciación, evitando pasos intermedios.

Todas las funciones de codificación beben de un módulo de lógica auxiliar llamado *Utils*, cuya única función es asistir en el manejo de estructuras de datos. Este módulo existe en ambos, cliente y servidor.

Una vez llega al servidor la petición, éste se encarga de separar el identificador y los parámetros. Comprueba el identificador y llama a la función de lógica correspondiente. Todas las funciones de lógica que se comunican directamente con el servidor decodifican los datos por sí mismas, por lo que sólo tiene que pasarle los parámetros que ha recibido.

Al terminar, si todo ha ido bien, se devolverá un mensaje OK junto con los datos que el cliente requieriera, de serel caso.. En caso de haber algún problema, recibirá un mensaje ERROR.

6.3. Cliente

La función del cliente es recibir datos del usuario para mandarlos al servidor, siendo su carga de procesamiento aquella relacionada con la interfaz gráfica y las comprobaciones de seguridad básica.

6.3.1. Flujo del juego

Ya que la función principal del cliente es la de servir de interfaz gráfica al usuario, comprender el flujo del juego resulta vital para comprender el funcionamiento del cliente. Se muestra a continuación la explicación de las diferentes pantallas del juego, como se explica en el Manual de Usuario[G], para mostrar en profundidad su implementación.

6.3.1.1. Identificación

Lo primero que se muestra al abrir la aplicación es la pantalla de identificación.

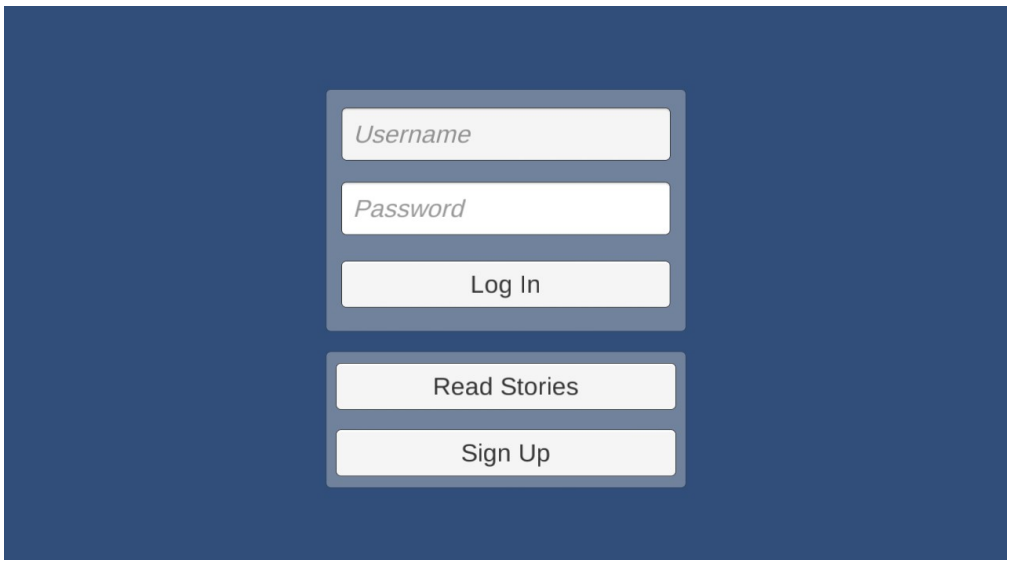


Figura 6.2 Pantalla de identificación

Aquí, el usuario puede *loggearse* utilizando su nombre de usuario y su contraseña, o registrarse presionando el botón *Sign Up* (Registrarse). El botón *Read Stories* (Leer historias) permitiría a un usuario anónimo leer historias de otros jugadores, pero se trata de una funcionalidad todavía no implementada.

6.3.1.2. Registro

Este es el panel de registro. Es el panel que un usuario utilizará para registrarse en la aplicación y, por tanto, tener acceso a todas sus funcionalidades.

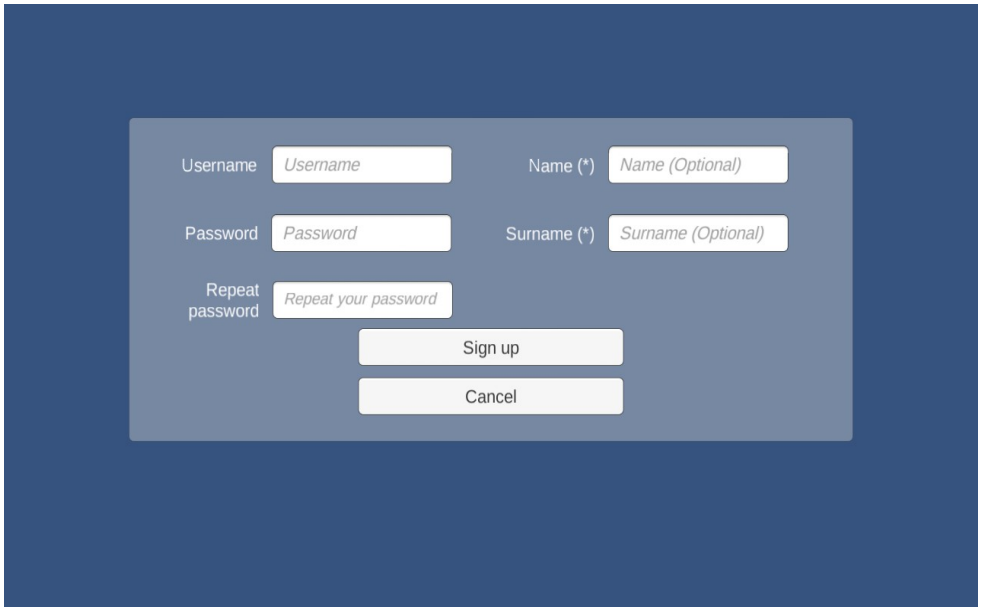


Figura 6.3 Pantalla de registro

Para registrarse, el usuario debe rellenar los campos en blanco junto a *Username* (Nombre de usuario) y *Password* (Contraseña), así como repetir la contraseña para asegurar que la contraseña era correcta. Los campos de Nombre de Usuario, Contraseña y Repetir contraseña son los únicos campos obligatorios para registrarse en el juego. Los campos *Name* (Nombre) y *Surname* (Apellido) son opcionales. De rellenarlos, esta información sólo estará disponible si se visita el perfil del jugador en la aplicación.

6.3.1.3. Panel de Usuario

Tras identificarse en la aplicación, se le presentará al usuario el Panel de Usuario. Éste es el cuartel general de jugador. Echémosle un vistazo:

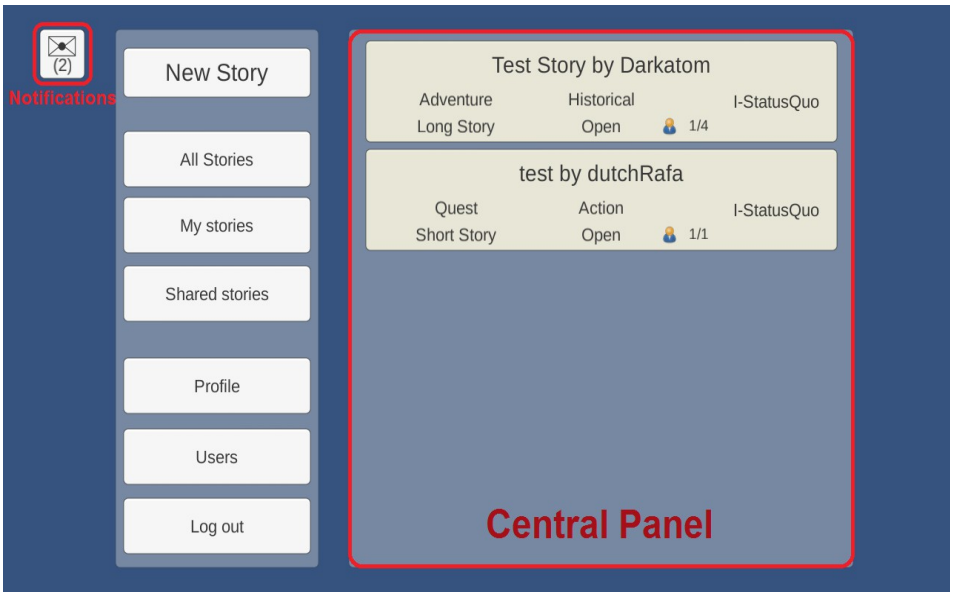


Figura 6.4 Panel de Usuario

Notificaciones: Las notificaciones son mensajes que el sistema manda al usuario sobre determinados eventos, como puede ser el haber perdido un turno enuna historia o que haya llegado el turno del usuario en otra.

Al pulsar el botón, se le mostrará al usuario un panel con todos los mensajes sin leer y un botón *Volver*. Al pulsar *Volver*, todos los mensajes son borrados y el usuario vuelve al panel de usuario.

Nueva Historia (New Story): Lleva al usuario al panel para crear una nueva historia.

Todas las Historias (All Stories): Enseña en el *panel central* (Central Panel) todas las historias en el sistema, ambas en progreso y finalizadas, aunque el jugador las haya abandonado. Tocar una llevará al jugador al *panel de juego*, donde podrá leer la historia y jugar un nuevo turno, si es que está disponible.

Mis Historias (My Stories): Enseña en el *panel central* todas las historias que el jugador esté jugando actualmente, ambas en progreso y finalizadas, aunque el jugador las haya abandonado. Tocar una llevará al jugador al *panel de juego*, donde podrá leer la historia y jugar un nuevo turno, si es que está disponible.

Historias Compartidas (Shared Stories): Enseña en el *panel central* todas las historias que el jugador esté jugando actualmente, tanto de las que es autor como las que no. Tocar una llevará al jugador al *panel de juego*, donde podrá leer la historia y jugar un nuevo turno, si es que está disponible.

Perfil (Profile): Se muestra en el *panel central* el perfil del jugador; su nombre, apellido, número de historias empezadas, número de historias jugadas, la media en porcentaje de turnos perdidos y la media en porcentaje de historias abandonadas. Estos datos se muestran para ayudar a otros jugadores a decidir si dejar que este jugador entre o no a su partida.

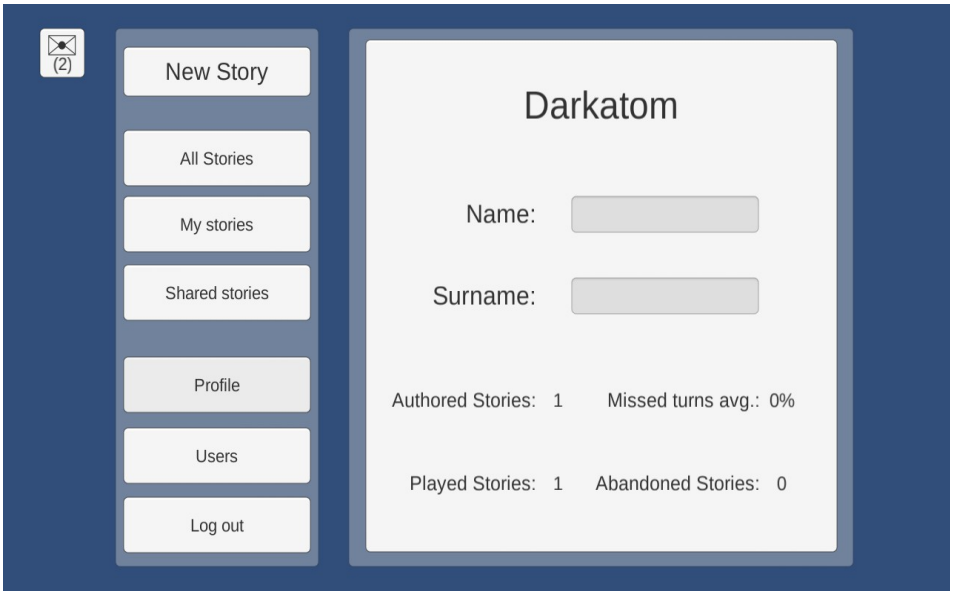


Figura 6.5 Perfil de un jugador

- **Usuarios (Users):** Enseña en el *panel central* la lista de usuarios registrados en el sistema. Tocar uno de ellos llevará al usuario al perfil de dicho jugador.

Figura 6.6 Lista de Usuarios

- **Desconectarse (Log Out):** Permite desconectarse del sistema.

6.3.1.4. Empezar una historia

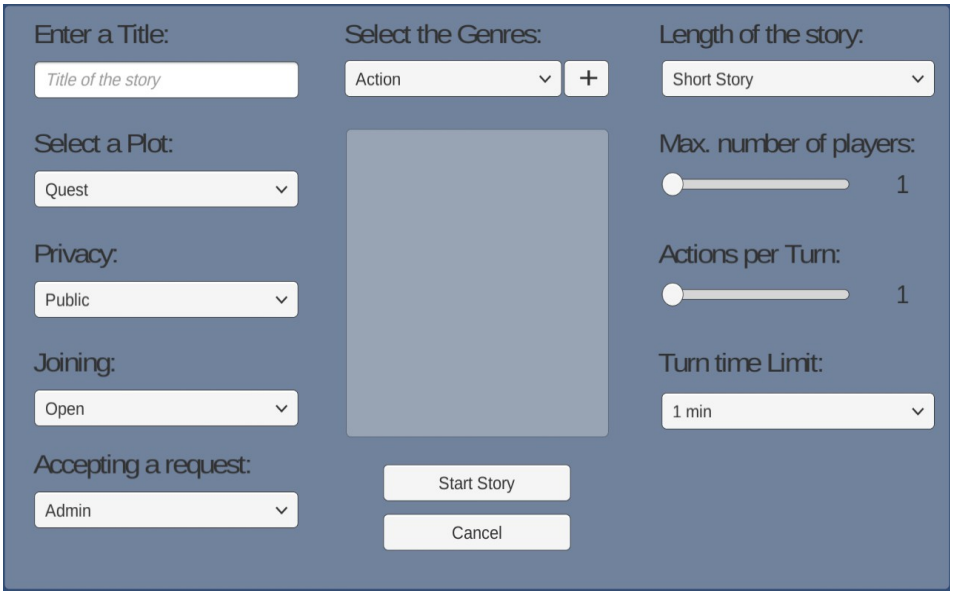


Figura 6.7 Panel de creación de historias

Este es el panel de creación de historias. Para minimizar los riesgos, se ha elegido que el único dato libre a introducir por los jugadores sea el título, siendo el resto opciones fijas. Estas opciones llegan del servidor y dependen únicamente de éste, de forma que si se quisieran añadir géneros de historia o tramas maestras al juego, el cliente no necesitaría ningún tipo de actualización.

El jugador seleccionará las opciones y los ajustes que desée para su historia antes de tocar el botón *Start Story* (Empezar Historia), que le llevará al panel de juego para introducir el primer turno. Introducir un título es obligatorio, aunque no se fuerza a que sea único en el sistema, por lo que no se permitirá continuar hasta que este campo sea rellenado.

6.3.1.5. Panel de Juego

En TaleBox, las historias son representadas como una serie de movimientos, desde el principio de la historia hasta su estado actual. Esto es lo que el jugador verá al entrar en el panel de juego (figura inferior), que muestra todos los movimientos jugados hasta el momento y un nuevo turno, si es que estuviera disponible.

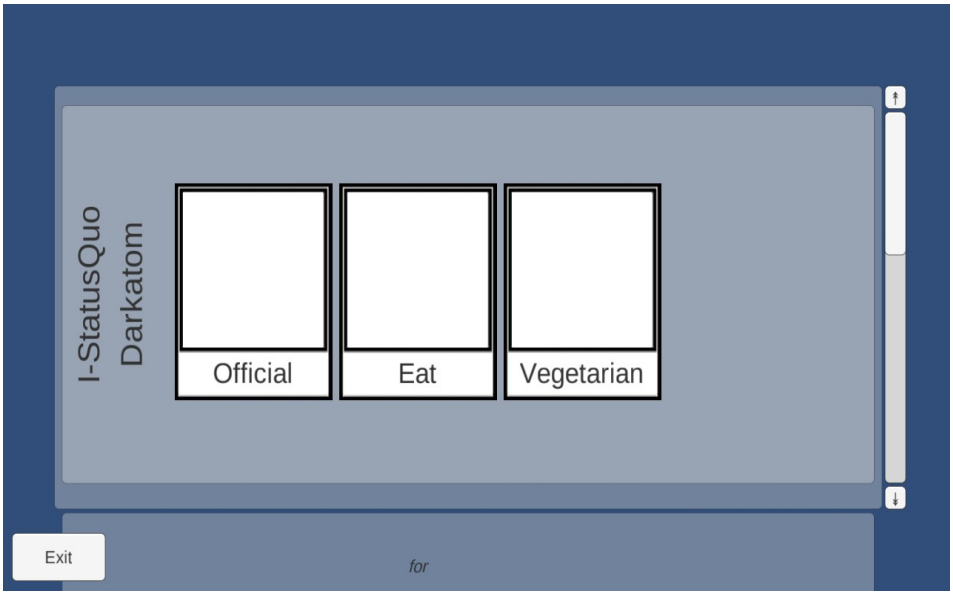


Figura 6.8 Representación de un movimiento en una historia

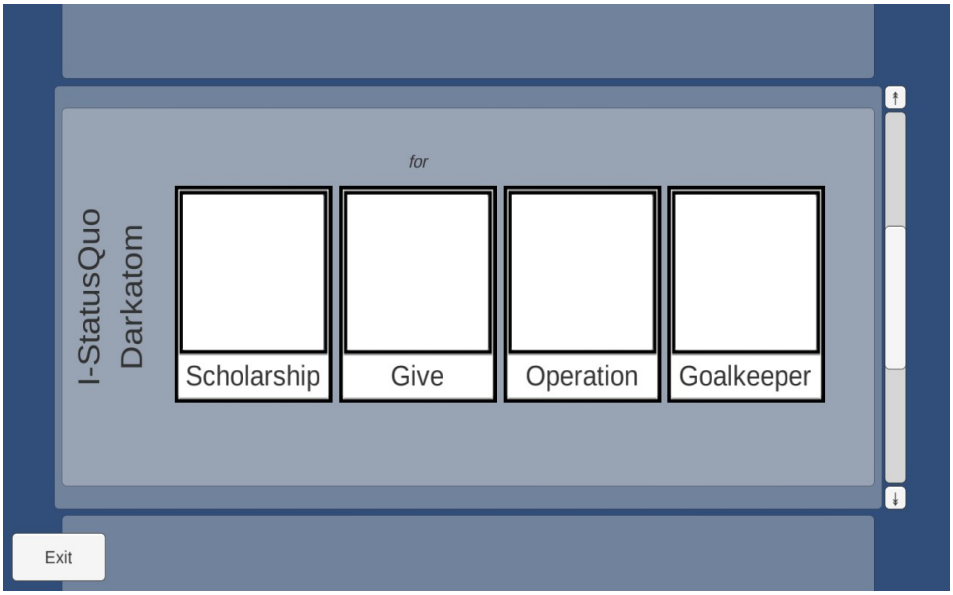


Figura 6.9 Otro movimiento

Los movimientos se representan como paneles que el jugador puede desplazar para leer la historia. La función de desplazamiento está disponible en todo momento, incluso mientras se rellena un turno, para que el jugador pueda consultar la historia en caso de duda.

Cada panel muestra el *Punto de Historia* al que pertenece el movimiento y el *Autor* del movimiento, es decir, el jugador que lo puso ahí. Las cartas cubren los cuatro elementos sintácticos descritos anteriormente; el sujeto, el verbo, el primer predicado y el segundo predicado, así como la/s preposición/preposiciones seleccionadas para el verbo.

Si el jugador tuviera un turno por jugar en la historia, se encontraría con el siguiente último panel:

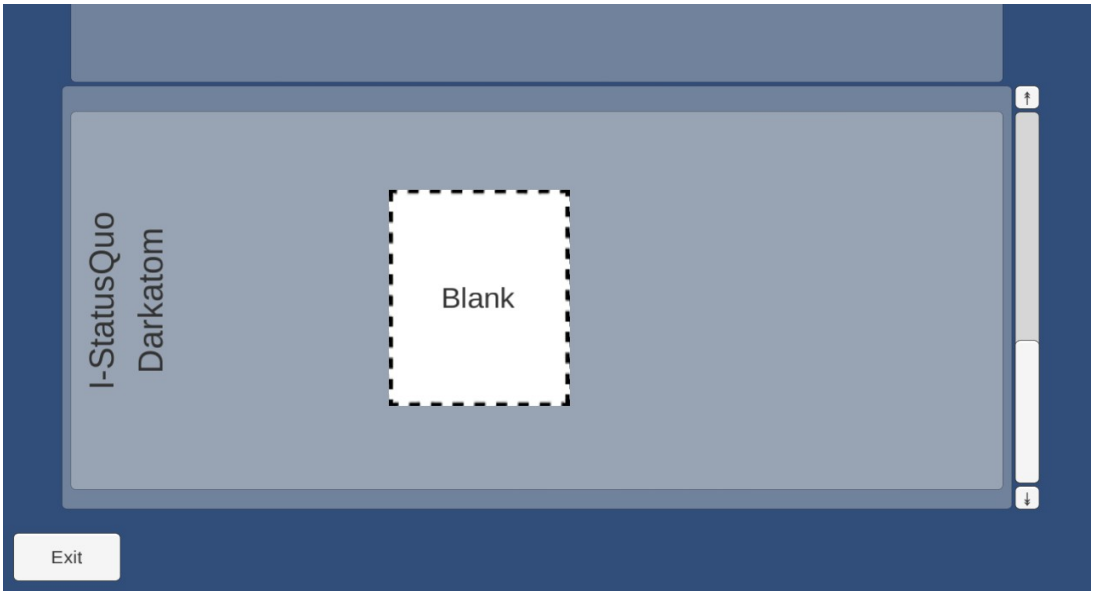


Figura 6.10 Movimiento en blanco – paso 1

Se trata de un movimiento en blanco.

Al haber entrado en la partida para leer la historia o poner su nuevo turno, el cliente ha recibido la mano que el jugador tiene para jugar en este turno. Las cartas son exclusivas de cada turno, y aquellas no utilizadas serán descartadas al terminar.

El primer blanco a rellenar es el verbo, siempre. El verbo determina el marco a rellenar en el movimiento, que a su vez determina qué elementos sintácticos participan y qué cartas podrán usarse en ellos.

Para rellenar el blanco basta con tocarlo, con lo que aparecerá una nueva lista de paneles. Esta vez, los paneles contienen las cartas disponibles. Tocar una la colocará en el hueco.

Tras elegir un verbo (en este ejemplo; *Linger*), aparecerán los blancos para formar una oración coherente con él (figura inferior). El número de espacios que aparecerán será igual al máximo que permitan los marcos del verbo, para la preposición escogida. En este caso, no hay preposiciones.

Es decir, si el verbo tuviera tres marcos, dos de tres espacios (sujeto, verbo y un predicado) y uno de cuatro (sujeto, verbo y los dos predicados), aparecían cuatro huecos a rellenar. Sin embargo, el cliente comprueba en todo momento si el movimiento es válido, por lo que con rellenar tres huecos correctamente sería sufcieinte para validar el movimiento.

De haber una preposición, aparecería la lista de posibles preposiciones encima de la carta *Verbo*. Estas preposiciones pueden ser vacías, simples (*con*) o compuestas (*de... para...*). Las preposiciones pueden aparecer repetidas en varios marcos del verbo, por lo que se aplicaría la regla anterior de posibilitar tantos huecos como tenga el marco más grande.

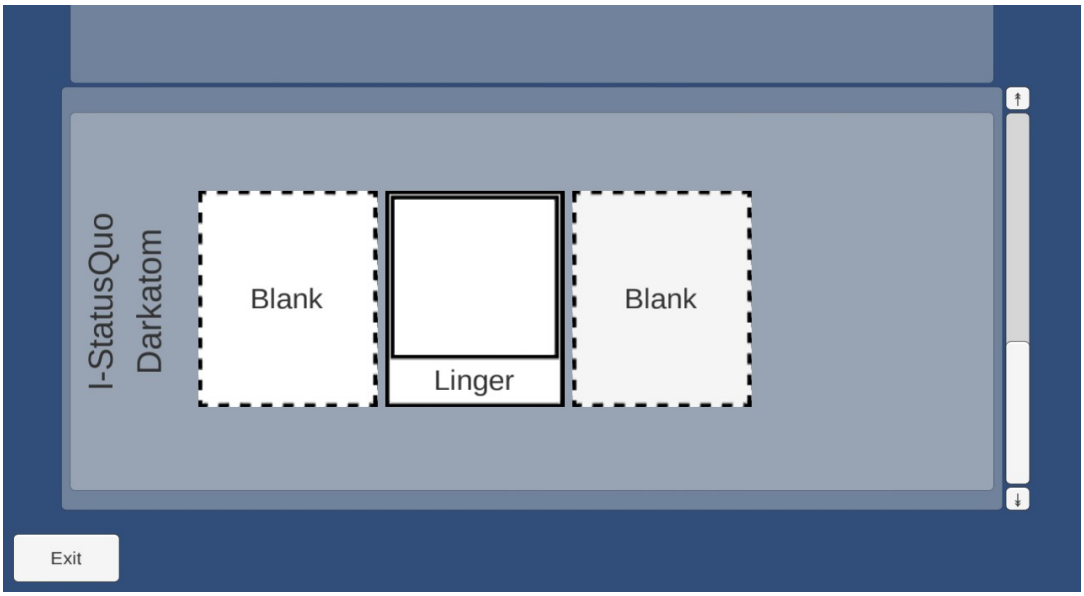


Figura 6.11 Movimiento en blanco – paso 2

Se deberán rellenar los huecos para obtener un movimiento válido.

Al entrar a elegir una carta para el hueco, el cliente automáticamente ofrece cartas que puedan rellenar dichos huecos a partir de los marcos que se activen con el estado actual del movimiento.

Es decir, el cliente sólo ofrecerá cartas que podrían validar el movimiento. Si se hubiera elegido un sujeto Abstracto, sólo ofrecerá cartas que concuerden con el hueco que se quiera rellenar en aquellos marcos cuyo sujeto pueda ser un elemento Abstracto.

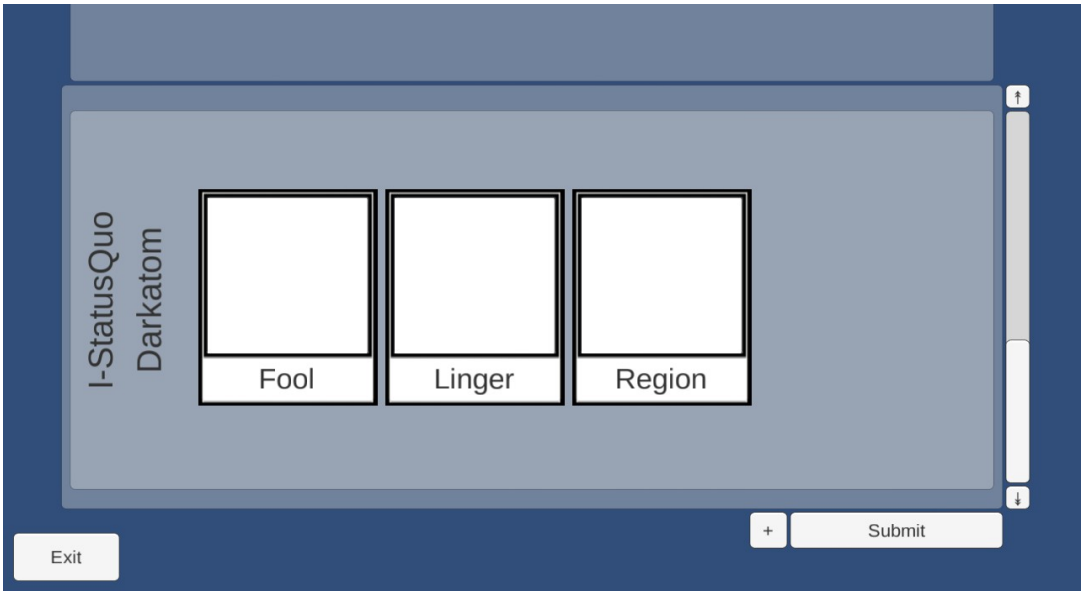


Figura 6.12 Movimiento en blanco – paso 3

Al rellenar un movimiento válido (en este caso, *a fool lingers in a region* – un tonto remolonea en una región) , aparecerán los botones *Añadir* y *Submit* (Mandar). *Añadir* creará un nuevo panel de turno vacío, y sólo aparecerá si todavía se pueden entregar más movimientos en este turno. *Mandar* recogerá todos los movimientos del turno, volverá a comprobar que sean correctos y los mandará al servidor para ser añadidos a la historia.

Una vez finalizado el turno, el jugador verá desaparecer los botones *Añadir* (si lo hubiera) y *Mandar*, mientras que sus movimientos permanecerán para ser leídos, ahora como parte de la historia. El siguiente jugador de la historia obtendrá una notificación de que su turno en la historia ha empezado, que recibirá al abrir su panel de usuario.

6.3.1.5. Implementación del *gameplay*

El servidor manda los movimientos sintácticamente al cliente, de forma que su único trabajo es mostrarlos correctamente.

Cada panel es un *prefab* (unidad prefabricada o modelo, en Unity) con una posición vacía para cada carta, que, a su vez, es otro *prefab*. El cliente colocará un nuevo panel *prefab* por cada movimiento que reciba. A la hora de colocar las cartas, éstas vienen en orden, por lo que sólo habrá de colocarlas en sus posiciones correspondientes, quedando aquellos elementos inexistentes vacíos. En caso de haber una preposición, se ha dispuesto una etiqueta encima del verbo para enseñarla, y que también quedará vacía en caso contrario.

Cada verbo viene con sus marcos, cada marco con su representación sintáctica y semántica, así como las restricciones semánticas de cada elemento sintáctico y las restricciones temáticas de cada restricción semántica. No todas las restricciones semánticas tienen restricciones temáticas; se puede dar el caso de que cualquier sustantivo valga para rellenar un campo semántico.

Los marcos se contrastan según avanza el turno, recortando las posibles cartas utilizables, llevando al jugador hacia movimientos válidos sin coartar su creatividad.

Las cartas seleccionadas pueden ser cambiadas en cualquier momento, incluyendo el verbo, que reiniciará todas las posiciones para evitar contradicciones con los marcos.

Una vez se elige mandar los movimientos al servidor, el cliente vuelve a comprobar que sean correctos y los formatea de acuerdo al pacto de comunicación que tenga con el servidor. Los manda y aguarda la confirmación, que puede llevar varios segundos. Una vez ha recibido una respuesta del servidor, los movimientos pasan a modo lectura. En modo lectura, tocar una carta no mostrará las cartas disponibles y la lista de preposiciones pasará a ser una etiqueta con la preposición que se eligió en el momento, sin posibilidad de cambiarla.

6.2.2. Gráficos

Los gráficos no eran parte del alcance de este proyecto. Sin embargo, se plantearon durante el transcurso de su desarrollo y se contró a una artista *freelance*, Sylvia Strijk, experimentada en ilustraciones y diseño gráfico, para encargarse del dibujado, el diseño de la interfaz gráfica, las cartas y otros elementos gráficos de la aplicación.

Los elementos gráficos de TaleBox debían ser simples y simbólicos. Un diseño demasiado serio podría llegar a desconectar a los jugadores de sus historias, o a influenciarlos hacia construcciones dramáticas. Un diseño demasiado infantil podría dar la impresión de que la aplicación no es seria y hacer que los jugadores pierdan interés por ella. Dada la gran influencia que los gráficos podrían tener en los jugadores, se decidió un enfoque a mitad de camino para

TaleBox; entre las ilustraciones y el diseño gráfico, de forma que su influencia sea neutra sin dejar de parecer seria.

Hay cuatro tipos de cartas en TaleBox: sustantivos, verbos, adjetivos y adverbios. Con estos cuatro elementos se puede formar cualquier movimiento en el juego. Cada tipo de carta, además, tiene un número de categorías, correspondientes a los roles temáticos que cubrirán los sustantivos, el tipo de verbo y la positividad o negatividad de los adjetivos y los adverbios.

Para facilitar la lectura en el juego de las cartas, se decidió que los tipos de cartas se representarían con colores, mientras que las categorías serían los símbolos en cada carta. Todos estos detalles se le entregaron a la artista en un documento llamado *Graphic Design Specifications*, el cuál se puede encontrar como anexo a este documento.

Aunque aún no se han podido implementar en TaleBox, estos son los diseños de algunas de las cartas a día de hoy:

Figura 6.13 Cartas de TaleBox

7

Conclusiones y futuro

7.1. Conclusiones

- Se ha diseñado e implementado StoryDB. La estructura es lo suficiente flexible para contener un gran número de historias y clasificarlas por trama y género.
- StoryDB guarda de forma coherente y legible sus historias. Esto facilita el análisis del espacio, la estructura, la tensión e intención de los personajes por medio de un computador, lo cuál facilitará el desarrollo software e investigaciones futuras en este campo.
- TaleBox es una herramienta que permite la inclusión de nuevas historias en StoryDB.
- TaleBox también es un videojuego de cartas, aún en desarrollo, que es capaz de componer manos de forma inteligente, adaptándose a las necesidades de la historia, sin dejar de ser aleatoria. Contiene, además, los ingredientes necesarios para atraer la atención del público *casual* y aquellos interesados en la creación de historias.
- La versatilidad de TaleBox fomentará la creación de nuevo contenido y ayudará a popular StoryDB con historias nuevas y originales.

De este proyecto ha salido un artículo corto que fue aprobado en a la conferencia DiGRA-FDG 2016 (http://digra-fdg2016.org/), para figurar como demo. El artículo puede encontrarse como anexo a esta memoria.

7.2. Futuro

Dado que TaleBox aún está en desarrollo, cabe esperar que en un futuro próximo se implemente:

- Una mejora gráfica considerable en la interfaz gráfica del usuario, incluyendo las cartas nuevas y una redistribución de las funcionalidades, a fin de que sean más intuitivas.
- Basándose en el ya implementado sistema de turnos, posibilitar el aspecto social del juego habilitando los pasos intermedios necesarios en el cliente para que los jugadores puedan unirse a las partidas.
- Habilitar la expulsión automática de jugadores que hayan perdido un número de turnos determinado.
- Habilitar los retos tras un turno, una característica vital para hacer de TaleBox divertido y atrayente a la comunidad de jugadores.
- Mejorar el sistema de reparto de cartas, teniendo en cuenta variables como el espacio y el punto de historia en el que se juega, a fin de ofrecer eventos y cartas más importantes dependiendo del esquema de trama en juego.

Por último, se espera que se dé una fase de encuesta en la satisfacción del jugador en la que se ajustará la aplicación, tras lo cuál, pasará a estar disponible en Google Play para dispositivos Android.

Bibliografía

[1]: *Detecting Story Analogies from Annotations of Time, Action and Agency*, David K. Elson.

[2]: *DramaBank: Annotating Agency in Narrative Discourse*, David K. Elson.

[3]: *A Tool for Deep Semantic Encoding of Narrative Texts*, David K. Elson and Kathleen R. McKeown.

[4]: *Building a Bank of Semantically Encoded Narratives*, David K. Elson and Kathleen R. McKeown.

[5]: *A Semantic Foundation for Mixed-Initiative Computational*, Storytelling, ICID's Interactive Storytelling 2015, Ben Kybartas and Rafael Bidarra.

[6]: Scrivener: *https://www.literatureandlatte.com/scrivener.php*

[7]: GluNet: *https://graphics.tudelft.nl/glunet/*

[8]: *20 Master Plots and How to Build Them, Ronald B. Tobias. ISBN:* 9780749914240

[9]: VerbNet: *http://verbs.colorado.edu/~mpalmer/projects/verbnet.html*

Anexo A: Análisis de Caperucita Roja

Este documento fue originalmente creado en una hoja de cálculo de Libre Office y ha sido impreso en formato PDF para facilitar la lectura de esta memoria.

Los documentos originales están disponibles para cualquiera que los solicite.

	TEXTUAL LAYER	TIMELINE LAYER		INTERPRETATIVE LAYER					
1	Once upon a time there lived in a certain village a little country girl, the prettiest creature who was ever seen. Her mother was excessively fond of her; and her grandmother doted on her still more. This good woman had a little red riding hood made for her. It suited the girl so extremely well that everybody called her Little Red Riding Hood.	ia →	**P: live(village, pretty(country(girl)))** **P: love(mom, girl)** **P: love(grandmother, girl)** **P: gift(mom, girl, hood)** **P: identity(girl, "Little Red Riding Hood")**						
2	One day her mother, having made some cakes, said to her, "Go, my dear, and see how your grandmother is doing, for I hear she has been very ill. Take her a cake, and this little pot of butter."	ia →	**P: bake(mom, cakes)** **P: give(mom, girl, basket(cake, butter))** **P: order(mom, girl, visit(girl, grandmother))** **P: order(mom, girl, give(girl, basket, grandmother))**	**implies GOAL: GIRL**	go(girl, house(grandmother))	**precondition for**	give(girl, grandmother, baske(cake, butter))	**provides for**	A: GRANDMOTHER
3	Little Red Riding Hood set out immediately to go to her grandmother, who lived in another village.	ia →	**P: live(other_village, grandmother)** **P: set[out](girl, visit(grandmother))**						
4	As she was going through the wood, she met with a wolf, who had a very great mind to eat her up, but he dared not, because of some woodcutters working nearby in the forest. He asked her where she was going. The poor child, who did not know that it was dangerous to stay and talk to a wolf, said to him, "I am going to see my grandmother and carry her a cake and a little pot of butter from my mother." "Does she live far off?" said the wolf "Oh I say," answered Little Red Riding Hood; "it is beyond that mill you see there, at the first house in the village." "Well," said the wolf, "and I'll go and see her too. I'll go this way and go you that, and we shall see who will be there first"	ia →	**P: encounter(girl, wolf)** **P: eat(wolf, girl) [1]** **P: fear(wolf, woodcutters) [2]** **P: ask(wolf, destination(girl))** **P: unkwon(girl, dangerous(wolf))** **P: tell(girl, wolf, destination(girl))** **P: challenge(wolf, race(girl))**	**[1] implies GOAL: WOLF** **[2] ceases GOAL: WOLF** **updates GOAL: WOLF**	eat(wolf, girl) slow[down](wolf, girl) ***precondition for*** go(wolf, destination(girl)) ***precondition for*** eat(wolf, grandmother) get[rid](wolf, woodcutters)	**provides for** **precondition for**	A: WOLF eat(wolf, girl)	**provides for**	A: WOLF
5	The wolf ran as fast as he could, taking the shortest path, and the little girl took a roundabout way, entertaining herself by gathering nuts, running after butterflies, and gathering bouquets of little flowers.	ia →	**P: run(fast(wolf), short_path)** **P: walked(entertained(girl), long_path)** **P: gather(girl, bouquet)**						
6	It was not long before the wolf arrived at the old woman's house. He knocked at the door: tap, tap. "Who's there?""Your grandchild, Little Red Riding Hood," replied the wolf, counterfeiting her voice; "who has brought you a cake and a little pot of butter sent you by mother. The good grandmother, who was in bed, because she was somewhat ill, cried out, "Pull the bobbin, and the latch will go up."	ia →	**P: arrive(wolf, house(grandmother))** **P: knock(wolf, door)** **P: ask(grandmother, identity(wolf))** **P: lie(wolf, identity(wolf))** **P: identity(wolf, girl)**						
7	The wolf pulled the bobbin, and the door opened, and then he immediately fell upon the good woman and ate her up in a moment, for it been more than three days since he had eaten. He then shut the door and got into the grandmother's bed, expecting Little Red Riding Hood, who came some time afterwards and knocked at the door: tap, tap.	ia →	**P: pull(wolf, bobbin)** **P: open(door)** **P: ate(wolf, grandmother)** **P: close(wolf, door)** **P: lie(wolf, bed(grandmother))** **P: identity(wolf, fake_grandmother)** **P: expect(wolf, girl)** **P: knock(girl, door)**	**updates GOAL: WOLF** **updates GOAL: GIRL**	deceive(wolf, girl) give(girl, grandmother, basket(cake, butter))	**precondition for** **provides for**	eat(wolf, girl) A: GRANDMOTHER	**provides for**	A: WOLF
8	"Who's there?" Little Red Riding Hood, hearing the big voice of the wolf, was at first afraid; but believing her grandmother had a cold and was hoarse, answered, "It is your grandchild Little Red Riding Hood, who has brought you a cake and a little pot of butter mother sends you." The wolf cried out to her, softening his voice as much as he could, "Pull the bobbin, and the latch will go up." Little Red Riding Hood pulled the bobbin, and the door opened.	ia →	**P: ask(wolf, identity(girl))** **P: hear(girl, voice(fake_grandmother))** **P: fear(girl, fake(grandmother))** **P: answer(girl, identity(girl)**						
9	The wolf, seeing her come in, said to her, hiding himself under the bedclothes, "Put the cake and the little pot of butter upon the stool, and come get into bed with me."	ia →	**P: order(fake_grandmother, put[upon](girl, basket(cake, butter), stool)** **P: order(fake_grandmother, get[into](girl, bed(fake_grandmother)))**	**updates GOAL: GIRL**	**B:** give(girl, grandmother, basket(cake, butter))	**provides for**	A: GRANDMOTHER		
10	Little Red Riding Hood took off her clothes and got into bed.	ia →	**P: take[off](girl, clothes(girl))** **P: get[into](girl, bed(fake_grandmother))**	**updates GOAL: WOLF**	eat(wolf, girl)	**provides for**	A: WOLF		
11	She was greatly amazed to see how her grandmother looked in her nightclothes, and said to her, "Grandmother, what big arms you have!" "All the better to hug you with, my dear." "Grandmother, what big legs you have!" "All the better to run with, my child." "Grandmother, what big ears you have!" "All the better to hear with, my child." "Grandmother, what big eyes you have!" "All the better to see with, my child." "Grandmother, what big teeth you have got!" "All the better to eat you up with." And, saying these words, this wicked wolf fell upon Little Red Riding Hood, and ate her all up.	ia →	**P: exclaim(girl, big(arms(fake_grandmother)))** **P: exclaim(girl, big(legs(fake_granmother)))** **P: exclaim(girl, big(ears(fake_grandmother)))** **P: exclaim(girl, big(eyes(fake_grandmother)))** **P: exclaim(girl, big(teeth(fake_grandmother)))** **P: eat(wolf, girl) [1]**	**[1] ceases GOAL: GIRL** **[1] success GOAL: WOLF**	give(girl, grandmother, basket(cake, butter))	**provides for**	A: GRANDMOTHER		

SCHEMA		PLOT REQUIREMENTS	CONCEPT	TEXTUAL LAYER	REPRESENTED AS	GluNET	STATE				FULFILLMENT
ACT I	STATUS QUO			Once upon a time there lived in a certain village a little country girl, the prettiest creature who was ever seen.	Girl is protagonist. Girl is from the country. Girl is pretty. Girl lives in country village. Girl's Mom loves Girl. Mom is secondary. Mom lives in country. Girl's Grandmother loves Girl. Grandmother is secondary. Grandmother gifts Girl a little red riding hood. Hood suits Girl. Girl is called Little Red Riding Hood.	[Person] Girl [Role] Protagonist [Attribute] Country [Attribute] Pretty [Place] Country [Place] Village [Person] Mom [Role] Secondary [Person] Grandmother [Item] Riding Hood [Attribute] Little [Attribute] Red [Identity] Little Red Riding Hood	**+ Village1:** **Role:** Setting **Attributes:** Country	**+ Girl1:** **Role**: Protagonist **Attributes:** Pretty Country **At:** Village1			
				Her mother was excessively fond of her; and her grandmother doted on her still more.			**+ RidingHood1:** **Role:** Item **Attributes:** Little Red	**+ Mom1:** **Role**: Secondary **At:** Village1 **Relationships:** [Mother to] Girl1	**Role**: Protagonist **Attributes:** Pretty Country **At:** Village1 **+ Relationships:** + [Daughter		
				This good woman had a little red riding hood made for her. It suited the girl so extremely well that everybody called her Little Red Riding Hood.				**+ Grandmother1:** **Role:** Secondary **Relationships:** [Grandmother to] Girl1	Country **At:** Village1 **+ Inventory:** RidingHood1 **+ Other identities:** + "Little Red Riding Hood" **Relationships:**		
	TRIGGER	Must have a ***motivating*** incident. Why does the hero want to go on the quest? **MUST:** Set the goal, "to find a place, a person or an object."	***Event***	One day her mother, having made some cakes, said to her, "Go, my dear, and see how your grandmother is doing, for I hear she has been very ill. Take her a cake, and this little pot of butter."	Mom bakes cakes. Mom orders Girl to [take cake and a pot of butter to Grandmother]. **Girl's Goal**: Find Grandmother, give her cake and butter.	[Event] Order [Item] Cake [Item] Pot of butter	**+ Cake1:** **Role:** Item **At:** Girl1 **+ PotOfButter1:** **Role:** Item **At:** Girl1	**At:** Village1 **Inventory:** RidingHood1 + Cake1 + PotOfButter1 **Other identities:** "Little Red	**+ Goal1:** **State:** Feasible **Steps:** Visit(Grandmother1) Give(Grandmother1, [Cake1, PotOfButter1])			**- [Event] ?** **[Event] Order** **- Goal Concept:** Find() **Protagonist Goals – Step == Goal Concept ?** Visit() == Find()
				Little Red Riding Hood set out immediately to go to her grandmother, who lived in another village.	Grandmother lives in another village.	[Attribute] Another [Place] Village	**+ Village2:** **Role:** Setting **Attributes:** Another	**Role**: Secondary **+ At:** Village2 **Relationships:**			
				As she was going through the wood, she met with a wolf, who had a very great mind to eat her up, but he dared not, because of some woodcutters working nearby in the forest.	Girl [starts to] go to Grandmother's house. Girl is in the woods. Girl meets Wolf. Wolf is antagonist. Wolf wants to eat Girl. Wolf can't eat Girl because [it's afraid of Woodcutters]. Woodcutters is secondary. Woodcutters is working in the forest. Woocutters is nearby Girl and Wolf.	[Place] Woods [Animal] Wolf [Role] Antagonist [Person] Woodcutter(s) [Obstacle] Afraid of Woodcutter(s) [Place] Forest -> Woods	**+ Woods1:** **Role:** Setting	**Inventory:** RidingHood1 Cake1 PotOfButter1 **Other identities:** "Little Red Riding Hood"	**+ Goal2:** **State:** Prevented **by** Woodcutters1 **Steps:** Remove(Woodcutters1) Eat(Girl1)	**Wolf1:** **Role:** Antagonist **Goal:** Goal2 **At**: Woods1 **Relationships:** [Scared of] Woodcutters1 [Acquaintance of]	**+ Woodcutters1:** **Role:** Secondary **At:** Woods1	

				He asked her where she was going. The poor child, who did not know that it was dangerous to stay and talk to a wolf, said to him, "I am going to see my grandmother and carry her a cake and a little pot of butter from my mother." "Does she live far off?" said the wolf "Oh I say," answered Little Red Riding Hood; "it is beyond that mill you see there, at the first house in the village."	Wolf asks Girl where is she going. Girl doesn't know Wolf is dangerous. Girl tells Wolf where she is going. Wolf learns Grandmother's living place.							
	FIRST PIVOT POINT			"Well," said the wolf, "and I'll go and see her too. I'll go this way and go you that, and we shall see who will be there first"	Wolf challenges Girl to a race.	[Event] Race			**Steps:** Visit(Grandmother1) **before** Girl1 Eat(Girl1) **Done:**			
				The wolf ran as fast as he could, taking the shortest path, and the little girl took a roundabout way, entertaining herself by gathering nuts, running after butterflies, and gathering bouquets of little flowers.	Wolf runs fast. Wolf takes shortest path to Grandmother's house. Girl takes long path to Grandmother's house. Girl entertains herself by [gathering nuts, running after butterflies, gathering bouquets of flowers.]	[Attribute] Fast [Attribute] Short [Place] Path [Attribute] Long [Event] Entertains -> Distracts -> Obstacle [Item] Nut(s) [Bug] Butterfly(s) [Item] Bouquet of flowers	**+ Nuts1:** **Role:** Item **At:** Girl1 **+ Bouquet1:** **Role:** Item **At:** Girl1 **+ Path1:** **Role:** Setting **Attributes:** Short **+ Path2:** **Role:** Setting **Attributes:** Long	**ry:** RidingHood1 Cake1 PotOfButter1 + Nuts1 + Bouquet1 **Other identities:** "Little Red Riding	**[U] Wolf1:** **Role:** Antagonist **Goal:** Goal2 + **Attributes:** + Fast **[U]At:** Path1 **Relationships:** [Scared of] Woodcutters1 [Acquaintance of] Girl1	**[U] Butterflies1:** **Role:** Secondary **At:** Path2		
				It was not long before the wolf arrived at the old woman's house. He knocked at the door: tap, tap. "Who's there?""Your grandchild, Little Red Riding Hood," replied the wolf, counterfeiting her voice; "who has brought you a cake and a little pot of butter sent you by mother. The good grandmother, who was in bed, because she was somewhat ill, cried out, "Pull the bobbin, and the latch will go up."	Wolf arrives at Grandmother's house. Wolf knocks at Grandmother's house door. Grandmother asks who's there. Wolf lies. Grandmother is ill. Grandmother is in bed. Grandmother believes Wolf is Girl. Wolf is Fake Girl. Grandmother tells Wolf how to enter. Grandmother commands Wolf [to enter].	[Place] House [Item] Door [State] Ill [Place] Bed [Identity] Fake Girl	**+ House1:** **Role:** Setting **At:** Village2 **Owner:** Grandmother1 **Inventory:** Door1 Bed1 **+ Door1:** **Role:** Item **At:** House1 **+ Bed1:** **Role:** Item **At:** House1	**Grandmother1:** **Role:** Secondary **[U]At:** House1 [Inside] **State:** Ill **Relationships:** [Grandmother	Goal2 **Attributes:** Fast **[U]At:** House1 [Outside] **+ Other identities:** + [Supplanting] Girl1 **Relationships:** [Scared of] Woodcutters1	Feasible **Steps:** + Eat(Grandmother1) Eat(Girl1) **Done:** Remove(Woodcutters1) + Visit(Grandmothe		
	MID-POINT			The wolf pulled the bobbin, and the door opened, and then he immediately fell upon the good woman and ate her up in a moment, for it been more than three days since he had eaten.	Wolf opens the door. Wolf enters Grandmother's house. Wolf is hungry. Wolf eats Grandmother. Grandmother is dead.	[State] Open [State] Hungry [State] Dead	**[U] Door1:** **Role:** Item **At:** House1 **+ State:** Open	**Role:** Secondary **At:** House1 **[U]State:** Dead **Relationships:** [Grandmother] Girl1	House1 [Inside] **+ State:** Hungry **Other identities:** [Supplanting] Girl1 **Relationships:** [Scared of]	**Steps:** Eat(Girl1) **Done:** Remove(Woodcutters1) Visit(Grandmother1) **before**	**[U] Goal1:** **State:** Prevented **by** Wolf1 **Steps:** Visit(Grandmother1) Give(Grandmother1, [Cake1, PotOfButter1])	

ACT II				He then shut the door and got into the grandmother's bed, expecting Little Red Riding Hood, who came some time afterwards and knocked at the door: tap, tap. "Who's there?"	Wolf shuts the door. Wolf gets into Grandmother's bed. Wolf waits for Girl. Girl arrives at Grandmother's house. Girl knocks at Grandmother's house. Wolf asks who's there.	[State] Closed [Place] Bed [Place] House	**[U] Door1:** **Role:** Item **At:** House1 **[U]State:** Closed	**[U]At:** House1 [Inside] + Bed1 [Inside] **+ State:** Hungry **Other identities:** [Suppla nting] Girl1 **Relationships:** -	House1 [Outside] **Inventory:** RidingHood 1 Cake1 PotOfButter 1 Nuts1 Bouquet1 **Other identities:** "Little Red Riding Hood" **Relationshi[s:**	**[U] Goal1:** **State:** Prevented **by** Wolf1 **Steps:** Give(Grandmother1, [Cake1, PotOfButter1]) **Done:** + Visit(Grandmother1)
				Little Red Riding Hood, hearing the big voice of the wolf, was at first afraid; but believing her grandmother had a cold and was hoarse, answered, "It is your grandchild Little Red Riding Hood, who has brought you a cake and a little pot of butter mother sends you." The wolf cried out to her, softening his voice as much as he could, "Pull the bobbin, and the latch will go up." Little Red Riding Hood pulled the bobbin, and the door opened.	Girl is afraid of Wolf's voice. Girl believes Wolf is Grandmother. Wolf is Fake Grandmother. Girl believes Grandmother's voice is hoarse. Wolf tells Girl how to enter. Wolf commands Girl [to enter].	[State] Afraid [Identity] Fake Grandmother [State] Open [State] With a Cold [Body] Voice [State] Hoarse	**[U] Door1:** **Role:** Item **At:** House1 **[U]State:** Open	House1 [Inside] Bed1 [Inside] **State:** Hungry **Other identities:** [Suppla nting] Girl1 + [Suppla nting] Grandm other1 **Relationships:**	**+ State:** Afraid **+ Believes:** + Belief1 + Belief2 **Inventory:** RidingHood 1 Cake1 PotOfButter 1 Nuts1 Bouquet1 **Other identities:** "Little Red Riding Hood"	**Object:** Grandmother1 **Attribute:** State **Is:** With a Cold **+** **Belief2:** **Object:** Grandmother1 **Attribute:** [Body] -> Voice **Is:** Hoarse **Because of:**
				The wolf, seeing her come in, said to her, hiding himself under the bedclothes, "Put the cake and the little pot of butter upon the stool, and come get into bed with me."	Girl enters Grandmother's house. Wolf commands Girl [to put upon the stool the cake and the pot of butter].	[Place] Stool	**[U] Cake1:** **Role:** Item **[U]At:** Stool [Upon] **[U] PotOfButter1:** **Role:** Item **[U]At:** Stool [Upon]	**Believes:** Belief1 Belief2 **Inventory:** RidingHood1 - Cake1 - PotOfButter1 Nuts1 Bouquet1 **Other identities:** "Little	**Relationshi** **[U] Goal1:** **State:** Prevented **by** Wolf1 **Done:** + Give(Grandmother1, [Cake1, PotOfButter1]) Visit(Grandmother1)	

	SECOND PIVOIT POINT			Little Red Riding Hood took off her clothes and got into bed.	Girl takes off her clothes. Girl gets into bed with Wolf.	[Item] Clothe(s) [Place] Bed	**+ Clothes1:** **Role:** Item	**State:** Afraid **Believes:** Belief1 Belief2 **Inventory:** RidingHood1 Nuts1 Bouquet1 **Other identities:**	
ACT III	**CLIMAX**	***Revelation.*** What is discovered is (usually) different of what was sought.	***Revelation***	She was greatly amazed to see how her grandmother looked in her nightclothes, and said to her, "Grandmother, what big arms you have!" "All the better to hug you with, my dear." "Grandmother, what big legs you have!" "All the better to run with, my child." "Grandmother, what big ears you have!" "All the better to hear with, my child." "Grandmother, what big eyes you have!" "All the better to see with, my child." "Grandmother, what big teeth you have got!" "All the better to eat you up with."	Girl thinks [Wolf's arms are big]. Girl thinks [Wolf's legs are big]. Girl thinks [Wolf's ears are big]. Girl thinks [Wolf's eyes are big]. Girl thinks [Wolf's teeth are big]. Girl discovers Grandmother is dead. Girl discovers [Wolf has deceived Girl]. Girl learns not to trust strangers.	[Attribute] Big [Body Part] Arm(s) [Body Part] Leg(s) [Body Part] Ear(s) [Body Part] Eye(s) [Body Part] Tooth(s) Learns -> Revelation [Trait] Not trust strangers	**+ Belief3:** **Object: Fake** Grandmother1 **Attribute:** [Body] -> Arms **Is:** Big **+ Belief4:** **Object: Fake** Grandmother1 **Attribute:** [Body] -> Legs **Is:** Big **+ Belief5:** **Object: Fake** Grandmother1 **Attribute:** [Body] -> Eyes **Is:** Big **+ Belief6:** **Object: Fake** Grandmother1 **Attribute:** [Body] -> Teeth **Is:** Big	**Traits:** + Not(Trust(Strangers)) **Believes:** Belief1 Belief2 + Belief3 + Belief4 + Belief5 + Belief6 **Inventory:** RidingHood1 Nuts1 Bouquet1	
	CLIMAX ENDING	May give up or complete the quest. The hero has ***changed***.	***Trait change***	And, saying these words, this wicked wolf fell upon Little Red Riding Hood, and ate her all up.	**Goal failed:** Grandmother has not recieved Girl's items. Wolf eats Girl. Girl is dead.	[State] Dead		Dead **Traits:** Not(Trust(Strangers)) **Believes:** Belief1 Belief2 Belief3 Belief4 Belief5 Belief6 **Inventory:** RidingHood1 Nuts1 Bouquet1	**[U] Goal1:** **[U]State:** Failed **Done:** Give(Grandmother1, [Cake1, PotOfButter1]) Visit(Grandmother1) **[U] Goal2:** **[U]State:** Completed **Steps:** **Done:** Remove(Woodcutters1) Visit(Grandmother1) **before** Girl1 Eat(Grandmother1) + Eat(Girl1)

Concept: Trait Change
Start Protagonist Trait != End Protagonist Traits ?

Trait: - **!= Traits:** Not(Trust(Strangers))

	TEXTUAL LAYER	TIMELINE LAYER		INTERPRETATIVE LAYER					
1	Once upon a time there lived in a certain village a little country girl, the prettiest creature who was ever seen. Her mother was excessively fond of her; and her grandmother doted on her still more. This good woman had a little red riding hood made for her. It suited the girl so extremely well that everybody called her Little Red Riding Hood.	ia →	**P: live(village, pretty(country(girl)))** **P: love(mom, girl)** **P: love(grandmother, girl)** **P: gift(mom, girl, hood)** **P: identity(girl, "Little Red Riding Hood")**						
2	One day her mother, having made some cakes, said to her, "Go, my dear, and see how your grandmother is doing, for I hear she has been very ill. Take her a cake, and this little pot of butter."	ia →	**P: bake(mom, cakes)** **P: give(mom, girl, basket(cake, butter))** **P: order(mom, girl, visit(girl, grandmother))** **P: order(mom, girl, give(girl, basket, grandmother))**	**implies GOAL: GIRL**	go(girl, house(grandmother))	**precondition for**	give(girl, grandmother, baske(cake, butter))	**provides for**	A: GRANDMOTHER
3	Little Red Riding Hood set out immediately to go to her grandmother, who lived in another village.	ia →	**P: live(other_village, grandmother)** **P: set[out](girl, visit(grandmother))**						
4	As she was going through the wood, she met with a wolf, who had a very great mind to eat her up, but he dared not, because of some woodcutters working nearby in the forest. He asked her where she was going. The poor child, who did not know that it was dangerous to stay and talk to a wolf, said to him, "I am going to see my grandmother and carry her a cake and a little pot of butter from my mother." "Does she live far off?" said the wolf "Oh I say," answered Little Red Riding Hood; "it is beyond that mill you see there, at the first house in the village." "Well," said the wolf, "and I'll go and see her too. I'll go this way and go you that, and we shall see who will be there first"	ia →	**P: encounter(girl, wolf)** **P: eat(wolf, girl) [1]** **P: fear(wolf, woodcutters) [2]** **P: ask(wolf, destination(girl))** **P: unkwon(girl, dangerous(wolf))** **P: tell(girl, wolf, destination(girl))** **P: challenge(wolf, race(girl))**	**[1] implies GOAL: WOLF** **[2] ceases GOAL: WOLF** **updates GOAL: WOLF**	eat(wolf, girl) slow[down](wolf, girl) ***precondition for*** go(wolf, destination(girl)) ***precondition for*** eat(wolf, grandmother) get[rid](wolf, woodcutters)	**provides for** **precondition for**	A: WOLF eat(wolf, girl)	**provides for**	A: WOLF
5	The wolf ran as fast as he could, taking the shortest path, and the little girl took a roundabout way, entertaining herself by gathering nuts, running after butterflies, and gathering bouquets of little flowers.	ia →	**P: run(fast(wolf), short_path)** **P: walked(entertained(girl), long_path)** **P: gather(girl, bouquet)**						
6	It was not long before the wolf arrived at the old woman's house. He knocked at the door: tap, tap. "Who's there?""Your grandchild, Little Red Riding Hood," replied the wolf, counterfeiting her voice; "who has brought you a cake and a little pot of butter sent you by mother. The good grandmother, who was in bed, because she was somewhat ill, cried out, "Pull the bobbin, and the latch will go up."	ia →	**P: arrive(wolf, house(grandmother))** **P: knock(wolf, door)** **P: ask(grandmother, identity(wolf))** **P: lie(wolf, identity(wolf))** **P: identity(wolf, girl)**						
7	The wolf pulled the bobbin, and the door opened, and then he immediately fell upon the good woman and ate her up in a moment, for it been more than three days since he had eaten. He then shut the door and got into the grandmother's bed, expecting Little Red Riding Hood, who came some time afterwards and knocked at the door: tap, tap.	ia →	**P: pull(wolf, bobbin)** **P: open(door)** **P: ate(wolf, grandmother)** **P: close(wolf, door)** **P: lie(wolf, bed(grandmother))** **P: identity(wolf, fake_grandmother)** **P: expect(wolf, girl)** **P: knock(girl, door)**	**updates GOAL: WOLF** **updates GOAL: GIRL**	deceive(wolf, girl) give(girl, grandmother, basket(cake, butter))	**precondition for** **provides for**	eat(wolf, girl) A: GRANDMOTHER	**provides for**	A: WOLF
8	"Who's there?" Little Red Riding Hood, hearing the big voice of the wolf, was at first afraid; but believing her grandmother had a cold and was hoarse, answered, "It is your grandchild Little Red Riding Hood, who has brought you a cake and a little pot of butter mother sends you." The wolf cried out to her, softening his voice as much as he could, "Pull the bobbin, and the latch will go up." Little Red Riding Hood pulled the bobbin, and the door opened.	ia →	**P: ask(wolf, identity(girl))** **P: hear(girl, voice(fake_grandmother))** **P: fear(girl, fake(grandmother))** **P: answer(girl, identity(girl)**						
9	The wolf, seeing her come in, said to her, hiding himself under the bedclothes, "Put the cake and the little pot of butter upon the stool, and come get into bed with me."	ia →	**P: order(fake_grandmother, put[upon](girl, basket(cake, butter), stool)** **P: order(fake_grandmother, get[into](girl, bed(fake_grandmother)))**	**updates GOAL: GIRL**	**B:** give(girl, grandmother, basket(cake, butter))	**provides for**	A: GRANDMOTHER		
10	Little Red Riding Hood took off her clothes and got into bed.	ia →	**P: take[off](girl, clothes(girl))** **P: get[into](girl, bed(fake_grandmother))**	**updates GOAL: WOLF**	eat(wolf, girl)	**provides for**	A: WOLF		
11	She was greatly amazed to see how her grandmother looked in her nightclothes, and said to her, "Grandmother, what big arms you have!" "All the better to hug you with, my dear." "Grandmother, what big legs you have!" "All the better to run with, my child." "Grandmother, what big ears you have!" "All the better to hear with, my child." "Grandmother, what big eyes you have!" "All the better to see with, my child." "Grandmother, what big teeth you have got!" "All the better to eat you up with." And, saying these words, this wicked wolf fell upon Little Red Riding Hood, and ate her all up.	ia →	**P: exclaim(girl, big(arms(fake_grandmother)))** **P: exclaim(girl, big(legs(fake_granmother)))** **P: exclaim(girl, big(ears(fake_grandmother)))** **P: exclaim(girl, big(eyes(fake_grandmother)))** **P: exclaim(girl, big(teeth(fake_grandmother)))** **P: eat(wolf, girl) [1]**	**[1] ceases GOAL: GIRL** **[1] success GOAL: WOLF**	give(girl, grandmother, basket(cake, butter))	**provides for**	A: GRANDMOTHER		

SCHEMA		PLOT REQUIREMENTS	CONCEPT	TEXTUAL LAYER	REPRESENTED AS	GluNET	STATE				FULFILLMENT
ACT I	STATUS QUO			Once upon a time there lived in a certain village a little country girl, the prettiest creature who was ever seen.	Girl is protagonist. Girl is from the country. Girl is pretty. Girl lives in country village. Girl's Mom loves Girl. Mom is secondary. Mom lives in country. Girl's Grandmother loves Girl. Grandmother is secondary. Grandmother gifts Girl a little red riding hood. Hood suits Girl. Girl is called Little Red Riding Hood.	[Person] Girl [Role] Protagonist [Attribute] Country [Attribute] Pretty [Place] Country [Place] Village [Person] Mom [Role] Secondary [Person] Grandmother [Item] Riding Hood [Attribute] Little [Attribute] Red [Identity] Little Red Riding Hood	**+ Village1:** **Role:** Setting **Attributes:** Country	**+ Girl1:** **Role**: Protagonist **Attributes:** Pretty Country **At:** Village1			
				Her mother was excessively fond of her; and her grandmother doted on her still more.			**+ RidingHood1:** **Role:** Item **Attributes:** Little Red	**+ Mom1:** **Role**: Secondary **At:** Village1 **Relationships:** [Mother to] Girl1	**Role**: Protagonist **Attributes:** Pretty Country **At:** Village1 **+ Relationships:** + [Daughter		
				This good woman had a little red riding hood made for her. It suited the girl so extremely well that everybody called her Little Red Riding Hood.				**+ Grandmother1:** **Role:** Secondary **Relationships:** [Grandmother to] Girl1	Country **At:** Village1 **+ Inventory:** RidingHood1 **+ Other identities:** + "Little Red Riding Hood" **Relationships:**		
	TRIGGER	Must have a ***motivating*** incident. Why does the hero want to go on the quest? **MUST:** Set the goal, "to find a place, a person or an object."	***Event***	One day her mother, having made some cakes, said to her, "Go, my dear, and see how your grandmother is doing, for I hear she has been very ill. Take her a cake, and this little pot of butter."	Mom bakes cakes. Mom orders Girl to [take cake and a pot of butter to Grandmother]. **Girl's Goal**: Find Grandmother, give her cake and butter.	[Event] Order [Item] Cake [Item] Pot of butter	**+ Cake1:** **Role:** Item **At:** Girl1 **+ PotOfButter1:** **Role:** Item **At:** Girl1	**At:** Village1 **Inventory:** RidingHood1 + Cake1 + PotOfButter1 **Other identities:** "Little Red	**+ Goal1:** **State:** Feasible **Steps:** Visit(Grandmother1) Give(Grandmother1, [Cake1, PotOfButter1])			**- [Event] ?** **[Event] Order** **- Goal Concept:** Find() **Protagonist Goals – Step == Goal Concept ?** Visit() == Find()
				Little Red Riding Hood set out immediately to go to her grandmother, who lived in another village.	Grandmother lives in another village.	[Attribute] Another [Place] Village	**+ Village2:** **Role:** Setting **Attributes:** Another	**Role**: Secondary **+ At:** Village2 **Relationships:**			
				As she was going through the wood, she met with a wolf, who had a very great mind to eat her up, but he dared not, because of some woodcutters working nearby in the forest.	Girl [starts to] go to Grandmother's house. Girl is in the woods. Girl meets Wolf. Wolf is antagonist. Wolf wants to eat Girl. Wolf can't eat Girl because [it's afraid of Woodcutters]. Woodcutters is secondary. Woodcutters is working in the forest. Woocutters is nearby Girl and Wolf.	[Place] Woods [Animal] Wolf [Role] Antagonist [Person] Woodcutter(s) [Obstacle] Afraid of Woodcutter(s) [Place] Forest -> Woods	**+ Woods1:** **Role:** Setting	**Inventory:** RidingHood1 Cake1 PotOfButter1 **Other identities:** "Little Red Riding Hood"	**+ Goal2:** **State:** Prevented **by** Woodcutters1 **Steps:** Remove(Woodcutters1) Eat(Girl1)	**Wolf1:** **Role:** Antagonist **Goal:** Goal2 **At**: Woods1 **Relationships:** [Scared of] Woodcutters1 [Acquaintance of]	**+ Woodcutters1:** **Role:** Secondary **At:** Woods1	

				He asked her where she was going. The poor child, who did not know that it was dangerous to stay and talk to a wolf, said to him, "I am going to see my grandmother and carry her a cake and a little pot of butter from my mother." "Does she live far off?" said the wolf "Oh I say," answered Little Red Riding Hood; "it is beyond that mill you see there, at the first house in the village."	Wolf asks Girl where is she going. Girl doesn't know Wolf is dangerous. Girl tells Wolf where she is going. Wolf learns Grandmother's living place.							
	FIRST PIVOT POINT			"Well," said the wolf, "and I'll go and see her too. I'll go this way and go you that, and we shall see who will be there first"	Wolf challenges Girl to a race.	[Event] Race			**Steps:** Visit(Grandmother1) **before** Girl1 Eat(Girl1) **Done:**			
				The wolf ran as fast as he could, taking the shortest path, and the little girl took a roundabout way, entertaining herself by gathering nuts, running after butterflies, and gathering bouquets of little flowers.	Wolf runs fast. Wolf takes shortest path to Grandmother's house. Girl takes long path to Grandmother's house. Girl entertains herself by [gathering nuts, running after butterflies, gathering bouquets of flowers.]	[Attribute] Fast [Attribute] Short [Place] Path [Attribute] Long [Event] Entertains -> Distracts -> Obstacle [Item] Nut(s) [Bug] Butterfly(s) [Item] Bouquet of flowers	**+ Nuts1:** **Role:** Item **At:** Girl1 **+ Bouquet1:** **Role:** Item **At:** Girl1 **+ Path1:** **Role:** Setting **Attributes:** Short **+ Path2:** **Role:** Setting **Attributes:** Long	**ry:** RidingHood1 Cake1 PotOfButter1 + Nuts1 + Bouquet1 **Other identities:** "Little Red Riding	**[U] Wolf1:** **Role:** Antagonist **Goal:** Goal2 + **Attributes:** + Fast **[U]At:** Path1 **Relationships:** [Scared of] Woodcutters1 [Acquaintance of] Girl1	**[U] Butterflies1:** **Role:** Secondary **At:** Path2		
				It was not long before the wolf arrived at the old woman's house. He knocked at the door: tap, tap. "Who's there?""Your grandchild, Little Red Riding Hood," replied the wolf, counterfeiting her voice; "who has brought you a cake and a little pot of butter sent you by mother. The good grandmother, who was in bed, because she was somewhat ill, cried out, "Pull the bobbin, and the latch will go up."	Wolf arrives at Grandmother's house. Wolf knocks at Grandmother's house door. Grandmother asks who's there. Wolf lies. Grandmother is ill. Grandmother is in bed. Grandmother believes Wolf is Girl. Wolf is Fake Girl. Grandmother tells Wolf how to enter. Grandmother commands Wolf [to enter].	[Place] House [Item] Door [State] Ill [Place] Bed [Identity] Fake Girl	**+ House1:** **Role:** Setting **At:** Village2 **Owner:** Grandmother1 **Inventory:** Door1 Bed1 **+ Door1:** **Role:** Item **At:** House1 **+ Bed1:** **Role:** Item **At:** House1	**Grandmother1:** **Role:** Secondary **[U]At:** House1 [Inside] **State:** Ill **Relationships:** [Grandmother	Goal2 **Attributes:** Fast **[U]At:** House1 [Outside] **+ Other identities:** + [Supplanting] Girl1 **Relationships:** [Scared of] Woodcutters1	Feasible **Steps:** + Eat(Grandmother1) Eat(Girl1) **Done:** Remove(Woodcutters1) + Visit(Grandmothe		
	MID-POINT			The wolf pulled the bobbin, and the door opened, and then he immediately fell upon the good woman and ate her up in a moment, for it been more than three days since he had eaten.	Wolf opens the door. Wolf enters Grandmother's house. Wolf is hungry. Wolf eats Grandmother. Grandmother is dead.	[State] Open [State] Hungry [State] Dead	**[U] Door1:** **Role:** Item **At:** House1 **+ State:** Open	**Role:** Secondary **At:** House1 **[U]State:** Dead **Relationships:** [Grandmother] Girl1	House1 [Inside] **+ State:** Hungry **Other identities:** [Supplanting] Girl1 **Relationships:** [Scared of]	**Steps:** Eat(Girl1) **Done:** Remove(Woodcutters1) Visit(Grandmother1) **before**	**[U] Goal1:** **State:** Prevented **by** Wolf1 **Steps:** Visit(Grandmother1) Give(Grandmother1, [Cake1, PotOfButter1])	

ACT II				He then shut the door and got into the grandmother's bed, expecting Little Red Riding Hood, who came some time afterwards and knocked at the door: tap, tap. "Who's there?"	Wolf shuts the door. Wolf gets into Grandmother's bed. Wolf waits for Girl. Girl arrives at Grandmother's house. Girl knocks at Grandmother's house. Wolf asks who's there.	[State] Closed [Place] Bed [Place] House	**[U] Door1:** **Role:** Item **At:** House1 **[U]State:** Closed	**[U]At:** House1 [Inside] + Bed1 [Inside] **+ State:** Hungry **Other identities:** [Suppla nting] Girl1 **Relationships:** -	House1 [Outside] **Inventory:** RidingHood 1 Cake1 PotOfButter 1 Nuts1 Bouquet1 **Other identities:** "Little Red Riding Hood" **Relationshi[s:**	**[U] Goal1:** **State:** Prevented **by** Wolf1 **Steps:** Give(Grandmother1, [Cake1, PotOfButter1]) **Done:** + Visit(Grandmother1)
				Little Red Riding Hood, hearing the big voice of the wolf, was at first afraid; but believing her grandmother had a cold and was hoarse, answered, "It is your grandchild Little Red Riding Hood, who has brought you a cake and a little pot of butter mother sends you." The wolf cried out to her, softening his voice as much as he could, "Pull the bobbin, and the latch will go up." Little Red Riding Hood pulled the bobbin, and the door opened.	Girl is afraid of Wolf's voice. Girl believes Wolf is Grandmother. Wolf is Fake Grandmother. Girl believes Grandmother's voice is hoarse. Wolf tells Girl how to enter. Wolf commands Girl [to enter].	[State] Afraid [Identity] Fake Grandmother [State] Open [State] With a Cold [Body] Voice [State] Hoarse	**[U] Door1:** **Role:** Item **At:** House1 **[U]State:** Open	House1 [Inside] Bed1 [Inside] **State:** Hungry **Other identities:** [Suppla nting] Girl1 + [Suppla nting] Grandm other1 **Relationships:**	**+ State:** Afraid **+ Believes:** + Belief1 + Belief2 **Inventory:** RidingHood 1 Cake1 PotOfButter 1 Nuts1 Bouquet1 **Other identities:** "Little Red Riding Hood"	**Object:** Grandmother1 **Attribute:** State **Is:** With a Cold **+** **Belief2:** **Object:** Grandmother1 **Attribute:** [Body] -> Voice **Is:** Hoarse **Because of:**
				The wolf, seeing her come in, said to her, hiding himself under the bedclothes, "Put the cake and the little pot of butter upon the stool, and come get into bed with me."	Girl enters Grandmother's house. Wolf commands Girl [to put upon the stool the cake and the pot of butter].	[Place] Stool	**[U] Cake1:** **Role:** Item **[U]At:** Stool [Upon] **[U] PotOfButter1:** **Role:** Item **[U]At:** Stool [Upon]	**Believes:** Belief1 Belief2 **Inventory:** RidingHood1 - Cake1 - PotOfButter1 Nuts1 Bouquet1 **Other identities:** "Little	**Relationshi** **[U] Goal1:** **State:** Prevented **by** Wolf1 **Done:** + Give(Grandmother1, [Cake1, PotOfButter1]) Visit(Grandmother1)	

	SECOND PIVOIT POINT			Little Red Riding Hood took off her clothes and got into bed.	Girl takes off her clothes. Girl gets into bed with Wolf.	[Item] Clothe(s) [Place] Bed	**+ Clothes1:** **Role:** Item	**State:** Afraid **Believes:** Belief1 Belief2 **Inventory:** RidingHood1 Nuts1 Bouquet1 **Other identities:**	
ACT III	**CLIMAX**	***Revelation.*** What is discovered is (usually) different of what was sought.	***Revelation***	She was greatly amazed to see how her grandmother looked in her nightclothes, and said to her, "Grandmother, what big arms you have!" "All the better to hug you with, my dear." "Grandmother, what big legs you have!" "All the better to run with, my child." "Grandmother, what big ears you have!" "All the better to hear with, my child." "Grandmother, what big eyes you have!" "All the better to see with, my child." "Grandmother, what big teeth you have got!" "All the better to eat you up with."	Girl thinks [Wolf's arms are big]. Girl thinks [Wolf's legs are big]. Girl thinks [Wolf's ears are big]. Girl thinks [Wolf's eyes are big]. Girl thinks [Wolf's teeth are big]. Girl discovers Grandmother is dead. Girl discovers [Wolf has deceived Girl]. Girl learns not to trust strangers.	[Attribute] Big [Body Part] Arm(s) [Body Part] Leg(s) [Body Part] Ear(s) [Body Part] Eye(s) [Body Part] Tooth(s) Learns -> Revelation [Trait] Not trust strangers	**+ Belief3:** **Object: Fake** Grandmother1 **Attribute:** [Body] -> Arms **Is:** Big **+ Belief4:** **Object: Fake** Grandmother1 **Attribute:** [Body] -> Legs **Is:** Big **+ Belief5:** **Object: Fake** Grandmother1 **Attribute:** [Body] -> Eyes **Is:** Big **+ Belief6:** **Object: Fake** Grandmother1 **Attribute:** [Body] -> Teeth **Is:** Big	**Traits:** + Not(Trust(Strangers)) **Believes:** Belief1 Belief2 + Belief3 + Belief4 + Belief5 + Belief6 **Inventory:** RidingHood1 Nuts1 Bouquet1	
	CLIMAX ENDING	May give up or complete the quest. The hero has ***changed***.	***Trait change***	And, saying these words, this wicked wolf fell upon Little Red Riding Hood, and ate her all up.	**Goal failed:** Grandmother has not recieved Girl's items. Wolf eats Girl. Girl is dead.	[State] Dead		Dead **Traits:** Not(Trust(Strangers)) **Believes:** Belief1 Belief2 Belief3 Belief4 Belief5 Belief6 **Inventory:** RidingHood1 Nuts1 Bouquet1	**[U] Goal1:** **[U]State:** Failed **Done:** Give(Grandmother1, [Cake1, PotOfButter1]) Visit(Grandmother1) **[U] Goal2:** **[U]State:** Completed **Steps:** **Done:** Remove(Woodcutters1) Visit(Grandmother1) **before** Girl1 Eat(Grandmother1) + Eat(Girl1)

Concept: Trait Change
Start Protagonist Trait != End Protagonist Traits ?

Trait: - **!= Traits:** Not(Trust(Strangers))

Anexo B: Análisis de La Hermana Zorro

Este documento fue originalmente creado en una hoja de cálculo de Libre Office y ha sido impreso en formato PDF para facilitar la lectura de esta memoria.

Los documentos originales están disponibles para cualquiera que los solicite.

	TEXTUAL LAYER		TIMELINE LAYER	INTERPRETATIVE LAYER							
1	A man had three sons and no daughter. He prayed for a daughter, even if she was a fox.	ia →	**P: have(man, oldest_son)** **P: have(man, second_son)** **P: have(man, young_son)** **P: pray(man, daughter)** **P: [not]matter(daughter, is(daughter, fox))**	implies **GOAL: MAN**	have(man, daughter)	**provides for**	A: MAN				
2	His wife gave birth to a daughter, but when the girl was six, one of their cows died every night.	ia →	**P: birth(wife(man), daughter)** **P: daughter(6)** **P: die(cow, every_night)**	fulfilled **GOAL: MAN**							
3	He set his oldest son to watch.	ia →	**P: set(man, watch(oldest_son, cows))**	implies **GOAL: MAN**	discover(man, why(die(cows))	**provides for**	A: MAN				
	The boy watched, and told him that his sister did it, by pulling the liver out of the cow and eating it. His father accused him of having fallen asleep and having a nightmare. He threw his son out.		**P: watch(oldest_son, cows)** **P: tell(oldest_son, man, kill(girl, cow, pull(girl, cow, liver)))** **P: tell(oldest_son, man, eat(girl, liver))** **P: accuse(man, oldest_son, have(oldest_son, nightmare))** **P: throw[out](man, oldest_son)**	implies **GOAL: OLDEST_SON** fulfilled **GOAL: OLDEST_SON**	watch(oldest_son, cows)	**precondition for**	discover(why(die(cows)))	**provides for**	A: MAN		
				prevented **GOAL: MAN**	own(man, belief)	**prevented**	discover(man, why(die(cows))	**provides for**	A: MAN		
4	The second son was set to watch over the cows, and nothing happened until the moon was full again, but then the sister struck, and the second son was also thrown out.	ia →	**P: set(man, watch(second_son, cows))** **P: full(moon)** **P: struck(daughter)** **P: throw[out](man, second_son)**	implies **GOAL: SECOND_SON** fulfilled **GOAL: SECOND_SON**	watch(second_son, cows)	**precondition for**	discover(why(die(cows)))	**provides for**	A: MAN		
				prevented **GOAL: MAN**	own(man, belief)	**prevented**	discover(man, why(die(cows))	**provides for**	A: MAN		
5	When the youngest son was set to watch, and claimed that their sister had gone to the outhouse, he claimed that the cows must have died from seeing the moon.	ia →	**P: set(man, watch(young_son, cows))** **P: claim(young_son, go(daughter, outhouse))** **P: claim(young_son, die(cows, see(moon)))**	implies **GOAL: YOUNG_SON** fulfilled **GOAL: YOUNG_SON**	watch(young_son, cows)	**precondition for**	discover(why(die(cows)))	**provides for**	A: MAN		
				prevented **GOAL: MAN**	lie(young_son) own(man, belief)	**prevented**	discover(man, why(die(cows))	**provides for**	A: MAN		
6	The older brothers wandered until they met a Buddhist monk, who sent them back with three magical bottles.	ia →	**P: wander([oldest_son, second_son])** **P: met([oldest_son, second_son], Buddhist_monk)** **P: give(Buddhist_monk, [oldest_son, second_son], [white_bottle, blue_bottle, red_bottle])** **P: send[back](Buddhist_monk, [oldest_son, second_son])**	implies **GOAL:** **OLDEST_SON** **SECOND_SON**	go[back]([oldest_son, second_son], home)	**provides for**	A: OLDEST_SON SECOND_SON				
7	They found their sister living alone; she told them their parents and brother had died, and implored them to stay.	ia →	**P: find([oldest_son, second_son], living(alone(daughter)))** **P: tell(daughter, [oldest_son, second_son], die([man, wife(man), young_son]))** **P: implore(daughter, [oldest_son, second_son], stay)**								
	Finally, she persuaded them to stay the night and somehow made a rich meal for them. In the night, the older brother was woken by the sounds of chewing.		**P: persuade(daughter, [oldest_son, second_son], stay(night))** **P: make(daughter, rich(meal))** **P: wake(oldest_son, chewing(sound))**								
	He rolled over, saw the meal, and realized that they had been eating corpses.		**P: roll[over](older_son)** **P: see(older_son, meal)** **P: realize(oldest_son, is(meal, corpses)**								
8	The sister stood over his dead brother, eating his liver. She told him that she needed only one more to become a human.	ia →	**P: stand[over](daughter, dead(second_son))** **P: eat(daughter, liver(second_son))** **P: tell(daughter, oldest_son, need(daughter, liver(1), become(daughter, human)))**	implies **GOAL:** **DAUGHTER**	eat(daughter, liver(oldest_son))	**provides for**	become(daughter, human)	**provides for**	A: DAUGHTER		
				implies **GOAL:** **OLDEST_SON**	escape(oldest_son, daughter)	**provides for**	A: OLDEST_SON				
9	He fled, throwing the white bottle behind him, and it became a thicket of thorns. As a fox, she made her way through it.	ia →	**P: fly(oldest_son)** **P: throw[behind](oldest_son, white_bottle))** **P: become(white_bottle, thicket(thorns))** **P: is(daughter, fox)** **P: make[through](daughter, thicket(thorns))**								
	He threw the blue bottle behind him, and trapped her in a river, but as a fox, she swam ashore.		**P: throw[behind](oldest_son, blue_bottle))** **P: become(white_bottle, river)** **P: trap(river, daughter)** **P: is(daughter, fox)** **P: swim(daughter, shore(river))**								
	He threw the red bottle behind, and she was trapped in fire. It burned her until she was no more than a mosquito.		**P: throw[behind](oldest_son, red_bottle))** **P: become(Red_bottle, fire)** **P: trap(fire, daughter)** **P: burn(fire, daughter)** **P: turn(fire, daughter, mosquito)**	**OLDEST_SON** prevented **GOAL:** **DAUGHTER**	turn(oldest_son, daughter, mosquito)	**prevented**	eat(daughter, liver(oldest_son))	**provides for**	become(daughter, human)	**provides for**	A: DAUGHTER
				fulfilled **GOAL:** **OLDEST_SON**	escape(oldest_son, daughter)	**provides for**	A: OLDEST_SON				

SCHEMA		PLOT REQUIREMENTS	CONCEPT	TEXTUAL LAYER	REPRESENTED AS	GluNET	STATE			
ACT I	STATUS QUO			A man had three sons and no daughter.	Man has three sons: first, second, third. Man is secondary. First son is antagonist. Second son is antagonist. Third son is antagonist.	[Person] Man [Person] Son [Relationship] Son -> [implies] Father -> [infered from Father] Brother/Sister [Attribute] First [Attribute] Second [Attribute] Third	**+ Man:** **Role**: Secondary **Relationships:** [Father] Son1 [Father] Son2 [Father] Son3	**+ Son1:** **Role**: Antagonist **Atributes:** First **Relationships:** [Son] Man [Brother] Son2 [Brother] Son3	**+ Son2:** **Role**: Antagonist **Atributes:** Second **Relationships:** [Son] Man [Brother] Son1 [Brother] Son3	**+ Son3:** **Role**: Antagonist **Atributes:** Third **Relationships:** [Son] Man [Brother] Son1 [Brother] Son2
				He prayed for a daughter, even if she was a fox. His wife gave birth to a daughter, ...	Man wants a daughter. **Man's goal:** To have daughter. He needs [to pray]. Man doesn't mind if future daughter is a fox. Man's wife gives birth to a daughter. Man's wife is secondary. Daughter is protagonist.	[Person] Daughter [Person] Wife [Relationship] Wife -> [implies] Husband [Relationship] Daughter -> [implies] Father -> [infered from Father] Brother/Sister [Verb] To give birth -> [inferes relationship] Mother	**[U] Man:** **Role**: Secondary **[U]Relationships:** [Father] Son1 [Father] Son2 [Father] Son3 + [Husband] Wife + [Father] Daughter	**+ Daughter:** **Role**: Protagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3	**+ Wife:** **Role**: Secondary **Relationships:** [Wife] Man [Mother] Daughter	**[U] Son1:** **Role**: Antagonist **Atributes:** First **[U]Relationships:** [Son] Man [Brother] Son2 [Brother] Son3 + [Brother] Daughter
	TRIGGER	The antagonist propels the protagonist towards the release (liberation from the curse).	A curse appears, represented as a goal for the protagonist.	..., but when the girl was six, one of their cows died every night.	Six years pass. Daughter is now six years old. Each night, a cow dies. **[Goal] Daughter is cursed.** **Man's goal:** Discover who kills cows at night.	[Time] Years [Attribute] Years -> [inferes] Age [Time] Night [Animal] Cow(s) [Event] Die -> [inferes] [State] Dead	**[U] Man:** **Role**: Secondary **Goal:** GoalMan **Relationships:** [Father] Son1 [Father] Son2 [Father] Son3 [Husband] Wife [Father] Daughter	**[U] Daughter:** **Role**: Protagonist + **Age:** 6 + **Goal:** GoalProtagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3	**+ GoalProtagonist:** **State:** Feasible **Role:** Curse **Other identities:** Curse	**+ GoalMan:** **State:** Feasible **Steps:** discover(who(kill(cows)), night)
		The Metamorph can't explain the reasons for the curse. We see it during the state of the curse.	Event with Protagonist as Subject	He set his oldest son to watch.	Man sets first son [to watch]. **First son's goal:** Discover who kills cows at night First son needs to [watch the cows].	[Event] Set someone -> [inferes] Order	**[U] Son1:** **Role**: Antagonist + **Goal:** GoalSon1 **Atributes:** First **Relationships:** [Son] Man [Brother] Son2 [Brother] Son3 [Brother] Daughter	**+ GoalSon1:** **State:** Feasible **Steps:** watch(cows) discover(who(kill(cows)), night)		
	FIRST PIVOT POINT			The boy watched, and told him that his sister did it, by pulling the liver out of the cow and eating it.	First son watchs the cows. First son learns about Daughter's curse. First son tells Man that [daughter pulls liver out of a cow] and [daughter eats the liver]. **First son's goal fulfilled.**	[Event] Watch [Event] Learn [Event] Tell [Event] Pull out *body part* -> [inferes] Kill [Body] Liver	**[U] GoalSon1:** **State:** Feasible **Steps:** discover(who(kill(cows)), night) **[U]Done:** + watch(cows)	**[U] GoalSon1:** **State:** Feasible **Steps:** **[U]Done:** watch(cows) **+** discover(who(kill(cows)), night)	**[U] GoalSon1:** **[U]State:** Fulfilled **Steps:** **Done:** watch(cows) discover(who(kill(cows)), night)	

ACT II				His father accused him of having fallen asleep and having a nightmare.	Man does not believe first son. Man accuses First son of [having fallen sleep] and [having a nightmare].	[Event] Not believe				
				He threw his son out.	Man throws first son out.	[Event] Throw out				
				The second son was set to watch over the cows, and nothing happened until the moon was full again, but then the sister struck, and the second son was also thrown out.	Second son is set [to watch over the cows]. **Second son's goal:** Discover who kills cows at night Second son watchs the cows. Daughter strucks when [the moon is full]. **Second's son goal fulfilled.** Man does not believe second son. Second son is thrown out.	[Event] Set someone -> [inferes] Order [Event] Watch [Time] When [the moon is full] [Event] Struck [Event] Not believe [Event] Throw out	**[U] Son2:** **Role**: Antagonist + **Goal:** GoalSon2 **Atributes:** First **Relationships:** [Son] Man [Brother] Son1 [Brother] Son3 [Brother] Daughter	**+ GoalSon2:** **State:** Feasible **Steps:** watch(cows) discover(who(kill(cows)), night)	**[U] GoalSon2:** **State:** Feasible **Steps:** discover(who(kill(cows)), night) **[U]Done:** + watch(cows)	**[U] GoalSon2:** **State:** Feasible **Steps:** **[U]Done:** watch(cows) **+** discover(who(kill(cows)), night)
				When the youngest son was set to watch, and claimed that their sister had gone to the outhouse, he claimed that the cows must have died from seeing the moon.	Third son is set [to watch]. **Third son's goal:** Discover who kills cows at night **Third son's goal fulfilled.** Third son claims that [daughter has gone to the outhouse] and [cows die because [cows look at the moon]]. **Man's goal fulfilled.**	[Event] Set someone -> [inferes] Order [Event] Watch [Event] Claim [Event] Die [Event] Look at [Event] Believe	**[U] Son3:** **Role**: Antagonist + **Goal:** GoalSon3 **Atributes:** First **Relationships:** [Son] Man [Brother] Son1 [Brother] Son2 [Brother] Daughter	**+ GoalSon3:** **State:** Feasible **Steps:** watch(cows) discover(who(kill(cows)), night)	**[U] GoalSon3:** **State:** Feasible **Steps:** discover(who(kill(cows)), night) **[U]Done:** + watch(cows)	**[U] GoalSon3:** **State:** Feasible **Steps:** **[U]Done:** watch(cows) **+** discover(who(kill(cows)), night)

	MID-POINT			The older brothers wandered until they met a Buddhist monk, who sent them back with three magical bottles.	First and second son wander. First son and second son meet a Buddhist monk. Buddhist monk is secondary. **Buddhist monk's goal:** To help first son. **Buddhist monk's goal:** To help second son. Buddhist monk gives First son and Second son a white bottle, a blue bottle and a red bottle. White bottle, blue bottle and red bottle are magical. Buddhist monk sends First son and Second son back.	[Event] Wander [Event] Meet -> [inferes] [Relationship] Acquaintance [Person] Monk [Attribute] Buddhist [Event] Give [Item] Bottle [Attribute] White [Attribute] Blue [Attribute] Red [Attribute] Magical [Event] Send someone back -> [inferes] Order	**+ WhiteBottle:** **Role**: Item **Object:** Bottle **Atributes:** White Magical **+ BlueBottle:** **Role**: Item **Object:** Bottle **Atributes:** Blue Magical **+ RedBottle:** **Role**: Item **Object:** Bottle **Atributes:** Red Magical	**+ Monk:** **Role**: Secondary **Goal:** Goal1Monk Goal2Monk **Atributes:** Buddhist **Relationships:** [Acquaintance] Son1 [Acquaintance] Son2	**[U] Son1:** **Role**: Antagonist **Atributes:** First **+ Inventory:** + WhiteBottle + BlueBottle + RedBottle **[U]Relationships:** [Son] Man [Brother] Son2 [Brother] Son3 [Brother] Daughter + [Acquaintance] Monk	**[U] Son2:** **Role**: Antagonist **Atributes:** First **+ Inventory:** + WhiteBottle + BlueBottle + RedBottle **Relationships:** [Son] Man [Brother] Son1 [Brother] Son3 [Brother] Daughter + [Acquaintance] Monk
		Evolving relationships between the antagonist and the metamorph. The characters will generally move toward each other emotionally.	Event involving Protagonist and Antagonist	They found their sister living alone;	First son and Second son find daughter living alone.	[Event] Find [Event] Living (alone)					
				she told them their parents and brother had died, and implored them to stay.	Daughter tells First son and Second son that [man, man's wife and third son have died].	[Event] Tell [Event] Die -> [inferes] [State] Dead	**[U] Man:** **Role**: Secondary **+ State:** Dead **Relationships:** [Father] Son1 [Father] Son2 [Father] Son3 [Husband] Wife [Father] Daughter	**[U] Wife:** **Role:** Secondary **+ State:** Dead **Relationships:** [Wife] Man [Mother] Daughter	**[U] Son3:** **Role**: Antagonist **+ State:** Dead **Atributes:** First **Relationships:** [Son] Man [Brother] Son1 [Brother] Son2 [Brother] Daughter		
	SECOND PIVOIT POINT			Finally, she persuaded them to stay the night and somehow made a rich meal for them.	Daughter persuades first son and second son to stay the night. Daughter makes rich meal for First son and Second son.	[Event] Persuade [Event] Stay [Time] Night [Item] Meal [Attribute] Rich	**+ Meal:** **Role**: Item **Object:** Bottle **Atributes:** Rich				
				In the night, the older brother was woken by the sounds of chewing.	It is night. First son wakes by sounds of chewing.	[Time] Night [Event] Wake					
				He rolled over, saw the meal, and realized that they had been eating corpses.	First son rolls over. First son sees the meal. First son realizes the meal was corpses.	[Event] Roll over [Event] See [Item] Corpse	**[U] Meal:** **Role**: Item **[U]Object:** Corpse **Atributes:** Rich				
				The sister stood over his dead brother, eating his liver.	Daughter stands over second son. Second son is dead. Daughter is eating second son's liver.	[Event] Stand over [State] Dead [Body] Liver	**[U] Son2:** **Role**: Antagonist **+ State:** Dead **Atributes:** First **Inventory:** WhiteBottle BlueBottle RedBottle **Relationships:** [Son] Man [Brother] Son1 [Brother] Son3 [Brother] Daughter [Acquaintance] Monk	**+ Goal2Monk:** **State:** Unfeasible **Steps:** help(Son2)			

ACT III				She told him that she needed only one more to become a human.	Daughter tells first son that [daughter needs one more liver [to become human]]. **Update Daughter's goal:** **Role:** Curse release Step[1]: eat(liver(1))	[Event] Tell	**[U] GoalProtagonist:** **State:** Feasible **Role:** Curse **+ Steps:** eat(liver(1)) become(human) **Other identities:** Curse			
	CLIMAX	The terms of the release are fulfilled by the antagonist and the protagonist is freed from the curse.	The antagonist fulfills the release for the curse.	He fled, throwing the white bottle behind him, and it became a thicket of thorns.	**First son's goal:** To escape daughter. First son runs away. First son throws the white bottle behind him. The white bottle becomes a thicket of thorns.	[Event] Run away [Event] Throw [Event] Become [Item] Thicket of thorns	**[U] WhiteBottle:** **[U]Role**: Location **[U]Form:** Thicket of thorns **Atributes:** White Magical	**[U] Son1:** **Role**: Antagonist **+ Goal:** Goal2Son1 **Atributes:** First **[U]Inventory:** - WhiteBottle BlueBottle RedBottle **Relationships:** [Son] Man [Brother] Son2 [Brother] Son3 [Brother] Daughter [Acquaintance] Monk	**+ Goal2Son1:** **State:** Feasible **Steps:** escape(Daughter)	
				As a fox, she made her way through it.	Daughter takes fox form. Daughter makes her way through the thicket of thorns. Daughter takes human form.	[Event] Take *something* form -> [inferes] become *something* -> [inferes] [State] fox [Item] Thicket (of thorns) [Event] Take *something* form -> [inferes] become *something* -> [inferes] [State] human	**[U] Daughter:** **Role**: Protagonist **Age:** 6 **+ Form:** Fox **+ Location:** WhiteBottle **Goal:** GoalProtagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3	**[U] Daughter:** **Role**: Protagonist **Age:** 6 **[U]Form:** Human **- Location:** WhiteBottle **Goal:** GoalProtagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3		
				He threw the blue bottle behind him, and trapped her in a river, but as a fox, she swam ashore.	First son throws the blue bottle behind him. The blue bottle becomes a river. Daughter takes fox form. Daughter swims to the river's shore. Daughter takes human form.	[Event] Throw [Event] Become [Event] Take *something* form -> [inferes] become *something* -> [inferes] [State] fox [Item] River [Event] Take *something* form -> [inferes] become *something* -> [inferes] [State] human	**[U] BlueBottle:** **[U]Role**: Location **[U]Form:** River **Atributes:** Blue Magical	**[U] Son1:** **Role**: Antagonist **Goal:** Goal2Son1 **Atributes:** First **[U]Inventory:** - BlueBottle RedBottle **Relationships:** [Son] Man [Brother] Son2 [Brother] Son3 [Brother] Daughter [Acquaintance] Monk	**[U] Daughter:** **Role**: Protagonist **Age:** 6 **[U]Form:** Fox **+ Location:** BlueBottle **Goal:** GoalProtagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3	**[U] Daughter:** **Role**: Protagonist **Age:** 6 **[U]Form:** Human **- Location:** BlueBottle **Goal:** GoalProtagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3
				He threw the red bottle behind, and she was trapped in fire.	First son throws the red bottle behind him. The red bottle becomes a fire. Daughter is trapped in fire. Fire burns daughter to death. Daughter is released by death. **Daughter's goal fulfilled.** **First son's goal fulfilled.** **Buddhist monk's goal fulfilled.**	[Event] Throw [Event] Become [Event] Take *something* form -> [inferes] become *something* -> [inferes] [State] fox [Item] Thicket (of thorns) [Event] Take *something* form -> [inferes] become *something* -> [inferes] [State] human [Event] Burn to death -> [inferes] [Event] Kill -> [inferes] [State] Dead [State] Released	**[U] RedBottle:** **[U]Role**: Location **[U]Form:** Fire **Atributes:** Red Magical	**[U] Son1:** **Role**: Antagonist **Goal:** Goal2Son1 **Atributes:** First **[U]Inventory:** - RedBottle **Relationships:** [Son] Man [Brother] Son2 [Brother] Son3 [Brother] Daughter [Acquaintance] Monk	**[U] Daughter:** **Role**: Protagonist **Age:** 6 **Form:** Human **+ State:** + Dead + Released **+ Location:** RedBottle **Goal:** GoalProtagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3	**+ Goal1Monk:** **State:** Fulfilled **Steps:** help(Son1)
	ENDING	The metamorph may either revert to the original state or die.		It burned her until she was no more than a mosquito.	From the ashes, a mosquito is born. Mosquito is secondary	[Location] Ashes [Event] Born [Insect] Mosquito	**+ Ashes:** **Role**: Location	**+ Mosquito:** **Role**: Secondary **Location:** Ashes		

		FULFILLMENT	TENSION GRAPH			
			+ Man (S) + Son1 (A) + Son2 (A) + Son3 (A)	**Man** Son1 +0 Son2 +0 Son3 +0 **Son2** Man +0 Son1 +0 Son3 +0 **Son1** Man +0 Son2 +0 Son3 +0 **Son3** Man +0 Son1 +0 Son2 +0	**Man** Son1 0 Son2 0 Son3 0 **Son2** Man 0 Son1 0 Son3 0 **Son1** Man 0 Son2 0 Son3 0 **Son3** Man 0 Son1 0 Son2 0
[U] Son2: **Role**: Antagonist **Atributes:** Second **[U]Relationships:** [Son] Man [Brother] Son1 [Brother] Son3 + [Brother] Daughter	**[U] Son3:** **Role**: Antagonist **Atributes:** Third **[U]Relationships:** [Son] Man [Brother] Son1 [Brother] Son2 + [Brother] Daughter		Man (S) Son1 (A) Son2 (A) Son3 (A) + Wife(S) + Daughter(P)	**Man** Son1 +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son2** Man +0 Son1 +0 Son3 +0 Wife +0 Daughter +0	**Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Wife** Son1 +0 Son2 +0 Son3 +0 Man +0 Daughter +0 **Daughter** Son1 +0 Son2 +0 Son3 +0 Man +0 Wife +0	**Man** Son1 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son1** Man 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son2** Man 0 Son1 0 Son3 0 Wife 0 Daughter 0	**Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0 **Wife** Son1 0 Son2 0 Son3 0 Man 0 Daughter 0 **Daughter** Son1 0 Son2 0 Son3 0 Man 0 Wife 0
		A curse appears, represented as a goal for the protagonist. Protagonist + [Goal] Role: Curse ??? **Yes**	Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P)	**Man** Son1 +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son2** Man +0 Son1 +0 Son3 +0 Wife +0 Daughter +0	**Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Wife** Son1 +0 Son2 +0 Son3 +0 Man +0 Daughter +0 **Daughter** Son1 +0 Son2 +0 Son3 +0 Man +0 Wife +0	**Man** Son1 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son1** Man 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son2** Man 0 Son1 0 Son3 0 Wife 0 Daughter 0	**Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0 **Wife** Son1 0 Son2 0 Son3 0 Man 0 Daughter 0 **Daughter** Son1 0 Son2 0 Son3 0 Man 0 Wife 0
		Event with Protagonist as Subject Protagonsit [Event] (something) ??? **Yes** Daughter [kills] cow	Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P)	**Man** - Son1 +0 - Son2 +0 - Son3 +0 - Wife +0 - Daughter +0 **Son1** - Man +0 - Son2 +0 - Son3 +0 - Wife +0 - Daughter +0 **Son2** - Man +0 - Son1 +0 - Son3 +0 - Wife +0 - Daughter +0	**Son3** - Man +0 - Son1 +0 - Son2 +0 - Wife +0 - Daughter +0 **Wife** - Son1 +0 - Son2 +0 - Son3 +0 - Man +0 - Daughter +0 **Daughter** - Son1 +0 - Son2 +0 - Son3 +0 - Man +0 - Wife +0	**Man** Son1 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son1** Man 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son2** Man 0 Son1 0 Son3 0 Wife 0 Daughter 0	**Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0 **Wife** Son1 0 Son2 0 Son3 0 Man 0 Daughter 0 **Daughter** Son1 0 Son2 0 Son3 0 Man 0 Wife 0
			Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Son1 | Learn Curse = 10 | Daughter	**Man** Son1 +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter -10 **Son2** Man +0 Son1 +0 Son3 +0 Wife +0 Daughter +0	**Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Wife** Son1 +0 Son2 +0 Son3 +0 Man +0 Daughter +0 **Daughter** Son1 +10 Son2 +0 Son3 +0 Man +0 Wife +0	**Man** Son1 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son1** Man 0 Son2 0 Son3 0 Wife 0 Daughter -10 **Son2** Man 0 Son1 0 Son3 0 Wife 0 Daughter 0	**Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0 **Wife** Son1 0 Son2 0 Son3 0 Man 0 Daughter 0 **Daughter** Son1 10 Son2 0 Son3 0 Man 0 Wife 0

			Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Man1 | Not Believe = 5 | Son1	**Man** Son1 +5 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Son1** Man -5 Son2 +0 Son3 +0 Wife +0 Daughter -10 **Wife** Son1 +0 Son2 +0 Son3 +0 Man +0 Daughter +0 **Son2** Man +0 Son1 +0 Son3 +0 Wife +0 Daughter +0 **Daughter** Son1 +10 Son2 +0 Son3 +0 Man +0 Wife +0	**Man** Son1 5 Son2 0 Son3 0 Wife 0 Daughter 0 **Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0 **Son1** Man -5 Son2 0 Son3 0 Wife 0 Daughter -10 **Wife** Son1 0 Son2 0 Son3 0 Man 0 Daughter 0 **Son2** Man 0 Son1 0 Son3 0 Wife 0 Daughter 0 **Daughter** Son1 +10 Son2 0 Son3 0 Man 0 Wife 0
			Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Man1 | Throw Out = 10 | Son1	**Man** Son1 +10 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Son1** Man -10 Son2 +0 Son3 +0 Wife +0 Daughter -10 **Wife** Son1 +0 Son2 +0 Son3 +0 Man +0 Daughter +0 **Son2** Man +0 Son1 +0 Son3 +0 Wife +0 Daughter +0 **Daughter** Son1 +10 Son2 +0 Son3 +0 Man +0 Wife +0	**Man** Son1 15 Son2 0 Son3 0 Wife 0 Daughter 0 **Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0 **Son1** Man -15 Son2 0 Son3 0 Wife 0 Daughter -10 **Wife** Son1 0 Son2 0 Son3 0 Man 0 Daughter 0 **Son2** Man 0 Son1 0 Son3 0 Wife 0 Daughter 0 **Daughter** Son1 10 Son2 0 Son3 0 Man 0 Wife 0
[U] GoalSon2: **[U]State:** Fulfilled **Steps:** **Done:** watch(cows) discover(who(kill(cows)), night)			Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Son2 | Learn Curse = 10 | Daughter Man1 | Not Believe = 5 | Son2 Man1 | Throw Out = 10 | Son2	**Man** Son1 +0 Son2 +5 +10 Son3 +0 Wife +0 Daughter +0 **Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Wife** Son1 +0 Son2 +0 Son3 +0 Man +0 Daughter +0 **Son2** Man -5 -10 Son1 +0 Son3 +0 Wife +0 Daughter -10 **Daughter** Son1 +0 Son2 +10 Son3 +0 Man +0 Wife +0	**Man** Son1 15 Son2 15 Son3 0 Wife 0 Daughter 0 **Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0 **Son1** Man -15 Son2 0 Son3 0 Wife 0 Daughter -10 **Wife** Son1 0 Son2 0 Son3 0 Man 0 Daughter 0 **Son2** Man -15 Son1 0 Son3 0 Wife 0 Daughter -10 **Daughter** Son1 10 Son2 10 Son3 0 Man 0 Wife 0
[U] GoalSon3: **[U]State:** Fulfilled **Steps:** **Done:** watch(cows) discover(who(kill(cows)), night)	**[U] GoalMan:** **[U]State:** Feasible **Steps:** discover(who(kill(cows)), night)		Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Son3 | Learn Curse = 10 | Daughter Son3 | Lie = 5 | Man	**Man** Son1 +0 Son2 +0 Son3 -5 Wife +0 Daughter +0 **Son3** Man +5 Son1 +0 Son2 +0 Wife +0 Daughter -10 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Wife** Son1 +0 Son2 +0 Son3 +0 Man +0 Daughter +0 **Son2** Man+0 Son1 +0 Son3 +0 Wife +0 Daughter +0 **Daughter** Son1 +0 Son2 +0 Son3 +10 Man +0 Wife +0	**Man** Son1 15 Son2 15 Son3 -5 Wife 0 Daughter 0 **Son3** Man 5 Son1 0 Son2 0 Wife 0 Daughter -10 **Son1** Man -15 Son2 0 Son3 0 Wife 0 Daughter -10 **Wife** Son1 0 Son2 0 Son3 0 Man 0 Daughter 0 **Son2** Man -15 Son1 0 Son3 0 Wife 0 Daughter -10 **Daughter** Son1 10 Son2 10 Son3 10 Man 0 Wife 0

+ Goal1Monk:
State: Feasible
Steps:
help(Son1)

+ Goal2Monk:
State: Feasible
Steps:
help(Son2)

	Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) + Monk (S)	**Man** Son1 +0 **Son3** Son2 +0 Man +0 Son3 +0 Son1 +0 Wife +0 Son2 +0 Daughter +0 Wife +0 Daughter +0 **Son1** Man +0 **Wife** Son2 +0 Son1 +0 Son3 +0 Son2 +0 Wife +0 Son3 +0 Daughter +0 Man +0 Monk +0 Daughter +0 **Son2** **Daughter** Man+0 Son1 +0 Son1 +0 Son2 +0 Son3 +0 Son3 +0 Wife +0 Man +0 Daughter +0 Wife +0 Monk +0	**Man** Son1 15 **Son3** Son2 15 Man 5 Son3 -5 Son1 0 Wife 0 Son2 0 Daughter 0 Wife 0 Daughter -10 **Son1** Man -15 **Wife** Son2 0 Son1 0 Son3 0 Son2 0 Wife 0 Son3 0 Daughter -10 Man 0 Monk 0 Daughter 0 **Son2** **Daughter** Man -15 Son1 10 Son1 0 Son2 10 Son3 0 Son3 10 Wife 0 Man 0 Daughter -10 Wife 0 Monk 0
Event involving Protagonist and Antagonist P [E] A Or A [E] P ??? **Yes** Daughter [persuades] Son1 Daughter [persuades] Son2	Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Monk (S)	**Man** Son1 +0 **Son3** Son2 +0 Man +0 Son3 +0 Son1 +0 Wife +0 Son2 +0 Daughter +0 Wife +0 Daughter +0 **Son1** Man +0 **Wife** Son2 +0 Son1 +0 Son3 +0 Son2 +0 Wife +0 Son3 +0 Daughter +0 Man +0 Monk +0 Daughter +0 **Son2** **Daughter** Man+0 Son1 +0 Son1 +0 Son2 +0 Son3 +0 Son3 +0 Wife +0 Man +0 Daughter +0 Wife +0 Monk +0	**Man** Son1 15 **Son3** Son2 15 Man 5 Son3 -5 Son1 0 Wife 0 Son2 0 Daughter 0 Wife 0 Daughter -10 **Son1** Man -15 **Wife** Son2 0 Son1 0 Son3 0 Son2 0 Wife 0 Son3 0 Daughter -10 Man 0 Monk 0 Daughter 0 **Son2** **Daughter** Man -15 Son1 10 Son1 0 Son2 10 Son3 0 Son3 10 Wife 0 Man 0 Daughter -10 Wife 0 Monk 0
	Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Monk (S)	**Man** Son1 +0 **Son3** Son2 +0 Man +0 Son3 +0 Son1 +0 Wife +0 Son2 +0 Daughter +0 Wife +0 Daughter +0 **Son1** Man +0 **Wife** Son2 +0 Son1 +0 Son3 +0 Son2 +0 Wife +0 Son3 +0 Daughter +0 Man +0 Monk +0 Daughter +0 **Son2** **Daughter** Man+0 Son1 +0 Son1 +0 Son2 +0 Son3 +0 Son3 +0 Wife +0 Man +0 Daughter +0 Wife +0 Monk +0	**Man** Son1 15 **Son3** Son2 15 Man 5 Son3 -5 Son1 0 Wife 0 Son2 0 Daughter 0 Wife 0 Daughter -10 **Son1** Man -15 **Wife** Son2 0 Son1 0 Son3 0 Son2 0 Wife 0 Son3 0 Daughter -10 Man 0 Monk 0 Daughter 0 **Son2** **Daughter** Man -15 Son1 10 Son1 0 Son2 10 Son3 0 Son3 10 Wife 0 Man 0 Daughter -10 Wife 0 Monk 0

The antagonist fulfills the release for the curse. A [Event] -> P [State] +Released ??? **Yes** Death -> Daughter [State] + Released Fire [Kill] Daughter RedBottle [turn] Fire Son1 [throw] RedBottle So Son1 [throw] RedBottle [turn] Fire [kill] Daughter ->Daughter [State] + Released			

[U] GoalProtagonist:
[U]State: Fulfilled
Role: Curse
Steps:
eat(liver(1))
become(human)
Other identities:
Curse

	TEXTUAL LAYER	TIMELINE LAYER		INTERPRETATIVE LAYER							
1	A man had three sons and no daughter. He prayed for a daughter, even if she was a fox.	ia →	**P: have(man, oldest_son)** **P: have(man, second_son)** **P: have(man, young_son)** **P: pray(man, daughter)** **P: [not]matter(daughter, is(daughter, fox))**	implies **GOAL: MAN**	have(man, daughter)	**provides for**	A: MAN				
2	His wife gave birth to a daughter, but when the girl was six, one of their cows died every night.	ia →	**P: birth(wife(man), daughter)** **P: daughter(S)** **P: die(cow, every_night)**	fulfilled **GOAL: MAN**							
3	He set his oldest son to watch.	ia →	**P: set(man, watch(oldest_son, cows))**	implies **GOAL: MAN**	discover(man, why(die(cows))	**provides for**	A: MAN				
	The boy watched, and told him that his sister did it, by pulling the liver out of the cow and eating it. His father accused him of having fallen asleep and having a nightmare. He threw his son out.		**P: watch(oldest_son, cows)** **P: tell(oldest_son, man, kill(girl, cow, pull(girl, cow, liver)))** **P: tell(oldest_son, man, eat(girl, liver))** **P: accuse(man, oldest_son, have(oldest_son, nightmare))** **P: throw[out](man, oldest_son)**	implies **GOAL: OLDEST_SON** fulfilled **GOAL: OLDEST_SON**	watch(oldest_son, cows)	**precondition for**	discover(why(die(cows)))	**provides for**	A: MAN		
				prevented **GOAL: MAN**	own(man, belief)	**prevented**	discover(man, why(die(cows))	**provides for**	A: MAN		
4	The second son was set to watch over the cows, and nothing happened until the moon was full again, but then the sister struck, and the second son was also thrown out.	ia →	**P: set(man, watch(second_son, cows))** **P: full(moon)** **P: struck(daughter)** **P: throw[out](man, second_son)**	implies **GOAL: SECOND_SON** fulfilled **GOAL: SECOND_SON**	watch(second_son, cows)	**precondition for**	discover(why(die(cows)))	**provides for**	A: MAN		
				prevented **GOAL: MAN**	own(man, belief)	**prevented**	discover(man, why(die(cows))	**provides for**	A: MAN		
5	When the youngest son was set to watch, and claimed that their sister had gone to the outhouse, he claimed that the cows must have died from seeing the moon.	ia →	**P: set(man, watch(young_son, cows))** **P: claim(young_son, go(daughter, outhouse))** **P: claim(young_son, die(cows, see(moon)))**	implies **GOAL: YOUNG_SON** fulfilled **GOAL: YOUNG_SON**	watch(young_son, cows)	**precondition for**	discover(why(die(cows)))	**provides for**	A: MAN		
				prevented **GOAL: MAN**	lie(young_son) own(man, belief)	**prevented**	discover(man, why(die(cows))	**provides for**	A: MAN		
6	The older brothers wandered until they met a Buddhist monk, who sent them back with three magical bottles.	ia →	**P: wander[oldest_son, second_son])** **P: met[(oldest_son, second_son), Buddhist_monk)** **P: give(Buddhist_monk, [oldest_son, second_son], [white_bottle, blue_bottle, red_bottle])** **P: send[back](Buddhist_monk, [oldest_son, second_son])**	implies **GOAL:** **OLDEST_SON** **SECOND_SON**	go[back]([oldest_son, second_son], home)	**provides for**	A: OLDEST_SON SECOND_SON				
7	They found their sister living alone; she told them their parents and brother had died, and implored them to stay.	ia →	**P: find[(oldest_son, second_son], living[alone(daughter)])** **P: tell(daughter, [oldest_son, second_son], die[man, wife(man), young_son])** **P: implore(daughter, [oldest_son, second_son], stay)**								
	Finally, she persuaded them to stay the night and somehow made a rich meal for them. In the night, the older brother was woken by the sounds of chewing.		**P: persuade(daughter, [oldest_son, second_son], stay(night))** **P: make(daughter, rich(meal))** **P: wake(oldest_son, chewing(sound))**								
	He rolled over, saw the meal, and realized that they had been eating corpses.		**P: roll[over](older_son)** **P: see(older_son, meal)** **P: realize(oldest_son, is(meal, corpses)**								
8	The sister stood over his dead brother, eating his liver. She told him that she needed only one more to become a human.	ia →	**P: stand[over](daughter, dead(second_son))** **P: eat(daughter, liver(second_son))** **P: tell(daughter, oldest_son, need(daughter, liver(1), become(daughter, human)))**	implies **GOAL:** **DAUGHTER**	eat(daughter, liver(oldest_son))	**provides for**	become(daughter, human)	**provides for**	A: DAUGHTER		
				implies **GOAL:** **OLDEST_SON**	escape(oldest_son, daughter)	**provides for**	A: OLDEST_SON				
9	He fled, throwing the white bottle behind him, and it became a thicket of thorns. As a fox, she made her way through it.	ia →	**P: fly(oldest_son)** **P: throw[behind](oldest_son, white_bottle))** **P: become(white_bottle, thicket(thorns))** **P: is(daughter, fox)** **P: make[through](daughter, thicket(thorns))**								
	He threw the blue bottle behind him, and trapped her in a river, but as a fox, she swam ashore.		**P: throw[behind](oldest_son, blue_bottle))** **P: become(white_bottle, river)** **P: trap(river, daughter)** **P: is(daughter, fox)** **P: swim(daughter, shore(river))**								
	He threw the red bottle behind, and she was trapped in fire. It burned her until she was no more than a mosquito.		**P: throw[behind](oldest_son, red_bottle))** **P: become(Red_bottle, fire)** **P: trap(fire, daughter)** **P: burn(fire, daughter)** **P: turn(fire, daughter, mosquito)**	OLDEST_SON prevented **GOAL:** **DAUGHTER**	turn(oldest_son, daughter, mosquito)	**prevented**	eat(daughter, liver(oldest_son))	**provides for**	become(daughter, human)	**provides for**	A: DAUGHTER
				fulfilled **GOAL:** **OLDEST_SON**	escape(oldest_son, daughter)	**provides for**	A: OLDEST_SON				

SCHEMA		PLOT REQUIREMENTS	CONCEPT	TEXTUAL LAYER	REPRESENTED AS	GIuNET	STATE						FULFILLMENT	TENSION GRAPH	
ACT I	STATUS QUO			A man had three sons and no daughter.	Man has three sons: first, second, third. Man is secondary. First son is antagonist. Second son is antagonist. Third son is antagonist.	[Person] Man [Person] Son [Relationship] Son -> [implies] Father -> [inferred from Father] Brother/Sister [Attribute] First [Attribute] Second [Attribute] Third	**+ Man:** **Role**: Secondary **Relationships:** [Father] Son1 [Father] Son2 [Father] Son3	**+ Son1:** **Role**: Antagonist **Atributes:** First **Relationships:** [Son] Man [Brother] Son2 [Brother] Son3	**+ Son2:** **Role**: Antagonist **Atributes:** Second **Relationships:** [Son] Man [Brother] Son1 [Brother] Son3	**+ Son3:** **Role**: Antagonist **Atributes:** Third **Relationships:** [Son] Man [Brother] Son1 [Brother] Son2				+ Man (S) + Son1 (A) + Son2 (A) + Son3 (A)	**Man** Son1 +0 Son2 +0 Son3 +0 **Son1** Man +0 Son2 +0 Son3 +0 **Son2** Man +0 Son1 +0 Son3 +0 **Son3** Man +0 Son1 +0 Son2 +0 **Man** Son1 0 Son2 0 Son3 0 **Son1** Man 0 Son2 0 Son3 0 **Son2** Man 0 Son1 0 Son3 0 **Son3** Man 0 Son1 0 Son2 0
				He prayed for a daughter, even if she was a fox. His wife gave birth to a daughter, ...	Man wants a daughter. **Man's goal:** To have daughter. He needs [to pray]. Man doesn't mind if future daughter is a fox. Man's wife gives birth to a daughter. Man's wife is secondary. Daughter is protagonist.	[Person] Daughter [Person] Wife [Relationship] Wife -> [implies] Husband [Relationship] Daughter -> [implies] Father -> [inferred from Father] Brother/Sister [Verb] To give birth -> [inferes relationship] Mother	**[U] Man:** **Role**: Secondary **[U]Relationships:** [Father] Son1 [Father] Son2 [Father] Son3 + [Husband] Wife + [Father] Daughter	**+ Daughter:** **Role**: Protagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3	**+ Wife:** **Role**: Secondary **Relationships:** [Wife] Man [Mother] Daughter	**[U] Son1:** **Role**: Antagonist **Atributes:** First **[U]Relationships:** [Son] Man [Brother] Son2 [Brother] Son3 + [Brother] Daughter	**[U] Son2:** **Role**: Antagonist **Atributes:** Second **[U]Relationships:** [Son] Man [Brother] Son1 [Brother] Son3 + [Brother] Daughter	**[U] Son3:** **Role**: Antagonist **Atributes:** Third **[U]Relationships:** [Son] Man [Brother] Son1 [Brother] Son2 + [Brother] Daughter		Man (S) Son1 (A) Son2 (A) Son3 (A) + Wife(S) + Daughter(P)	**Man** Son1 +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son2** Man +0 Son1 +0 Son3 +0 Wife +0 Daughter +0 **Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Man** Son1 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son1** Man 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son2** Man 0 Son1 0 Son3 0 Wife 0 Daughter 0 **Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0
	TRIGGER	The antagonist propels the protagonist towards the release (liberation from the curse).	A curse appears, represented as a goal for the protagonist.	..., but when the girl was six, one of their cows died every night.	Six years pass. Daughter is now six years old. Each night, a cow dies. **[Goal] Daughter is cursed.** **Man's goal:** Discover who kills cows at night.	[Time] Years [Attribute] Years -> [inferes] Age [Time] Night [Animal] Cow(s) [Event] Die -> [inferes] [State] Dead	**[U] Man:** **Role**: Secondary **Goal**: GoalMan **Relationships:** [Father] Son1 [Father] Son2 [Father] Son3 [Husband] Wife [Father] Daughter	**[U] Daughter:** **Role**: Protagonist + **Age:** 6 + **Goal**: GoalProtagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3	**+ GoalProtagonist:** **State:** Feasible **Role:** Curse **Other identities:** Curse	**+ GoalMan:** **State:** Feasible **Steps:** discover(who(kill(cows)), night)		A curse appears, represented as a goal for the protagonist. Protagonist + [Goal] Role: Curse ??? **Yes**	Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P)	**Man** Son1 +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son2** Man +0 Son1 +0 Son3 +0 Wife +0 Daughter +0 **Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Man** Son1 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son1** Man 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son2** Man 0 Son1 0 Son3 0 Wife 0 Daughter 0 **Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0
		The Metamorph can't explain the reasons for the curse. We see it during the state of the curse.	Event with Protagonist as Subject	He set his oldest son to watch.	Man sets first son [to watch]. **First son's goal:** Discover who kills cows at night First son needs to [watch the cows].	[Event] Set someone -> [inferes] Order	**[U] Son1:** **Role**: Antagonist + **Goal**: GoalSon1 **Atributes:** First **Relationships:** [Son] Man [Brother] Son2 [Brother] Son3 [Brother] Daughter	**+ GoalSon1:** **State:** Feasible **Steps:** watch(cows) discover(who(kill(cows)), night)				Protagonsit [Event] (something) ??? **Yes** Daughter [kills] cow	Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P)	**Man** Son1 +0 - Son2 +0 - Son3 +0 - Wife +0 - Daughter +0 **Son1** - Man +0 - Son2 +0 - Son3 +0 - Wife +0 - Daughter +0 **Son2** - Man +0 - Son1 +0 - Son3 +0 - Wife +0 - Daughter +0 **Son3** - Man +0 - Son1 +0 - Son2 +0 - Wife +0 - Daughter +0 **Man** Son1 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son1** Man 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son2** Man 0 Son1 0 Son3 0 Wife 0 Daughter 0 **Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0
	FIRST PIVOT POINT			The boy watched, and told him that his sister did it, by pulling the liver out of the cow and eating it.	First son watchs the cows. First son learns about Daughter's curse. First son tells Man that [daughter pulls liver out of a cow] and [daughter eats the liver]. **First son's goal fulfilled.**	[Event] Watch [Event] Learn [Event] Tell [Event] Pull out *body part* -> [inferes] Kill [Body] Liver	**[U] GoalSon1:** **State:** Feasible **Steps:** discover(who(kill(cows)), night) **[U]Done:** + watch(cows)	**[U] GoalSon1:** **State:** Feasible **Steps:** watch(cows) **[U]Done:** + discover(who(kill(cows)), night)	**[U] GoalSon1:** **[U]State:** Fulfilled **Steps:** watch(cows) discover(who(kill(cows)), night) **Done:**					Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Son1 | Learn Curse = 10 | Daughter	**Man** Son1 +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter -10 **Son2** Man +0 Son1 +0 Son3 +0 Wife +0 Daughter +0 **Son3** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Man** Son1 0 Son2 0 Son3 0 Wife 0 Daughter 0 **Son1** Man 0 Son2 0 Son3 0 Wife 0 Daughter -10 **Son2** Man 0 Son1 10 Son3 0 Wife 0 Daughter 0 **Son3** Man 0 Son2 0 Son3 0 Wife 0 Daughter 0
				His father accused him of having fallen asleep and having a nightmare.	Man does not believe first son. Man accuses First son of [having fallen sleep] and [having a nightmare].	[Event] Not believe								Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Man1 | Not Believe = 5 | Son1	**Man** Son1 +5 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son1** Man -5 Son2 +0 Son3 +0 Wife +0 Daughter -10 **Son2** Man +0 Son1 +0 Son3 +0 Wife +0 Daughter +0 **Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Man** Son1 5 Son2 0 Son3 0 Wife 0 Daughter 0 **Son1** Man -5 Son2 0 Son3 0 Wife 0 Daughter -10 **Son2** Man 0 Son1 10 Son3 0 Wife 0 Daughter 0 **Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0
				He threw his son out.	Man throws first son out.	[Event] Throw out								Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Man1 | Throw Out = 10 | Son1	**Man** Son1 +10 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son1** Man -10 Son2 +0 Son3 +0 Wife +0 Daughter -10 **Son2** Man +0 Son1 +0 Son3 +0 Wife +0 Daughter +0 **Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Man** Son1 15 Son2 0 Son3 0 Wife 0 Daughter 0 **Son1** Man -15 Son2 0 Son3 0 Wife 0 Daughter -10 **Son2** Man 0 Son1 10 Son3 0 Wife 0 Daughter 0 **Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0
				The second son was set to watch over the cows, and nothing happened until the moon was full again, but then the sister struck, and the second son was also thrown out.	Second son is set [to watch over the cows]. **Second son's goal:** Discover who kills cows at night Second son watchs the cows. Daughter strucks when [the moon is full]. **Second's son goal fulfilled.** Man does not believe second son. Second son is thrown out.	[Event] Set someone -> [inferes] Order [Event] Watch [Time] When [the moon is full] [Event] Struck [Event] Not believe [Event] Throw out	**[U] Son2:** **Role**: Antagonist + **Goal**: GoalSon2 **Atributes:** First **Relationships:** [Son] Man [Brother] Son1 [Brother] Son3 [Brother] Daughter	**+ GoalSon2:** **State:** Feasible **Steps:** watch(cows) discover(who(kill(cows)), night)	**[U] GoalSon2:** **State:** Feasible **Steps:** discover(who(kill(cows)), night) **[U]Done:** + watch(cows)	**[U] GoalSon2:** **State:** Feasible **Steps:** watch(cows) + discover(who(kill(cows)), night)	**[U] GoalSon2:** **[U]State:** Fulfilled **Steps:** watch(cows) discover(who(kill(cows)), night) **Done:**			Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Son2 | Learn Curse = 10 | Daughter Man1 | Not Believe = 5 | Son2 Man1 | Throw Out = 10 | Son2	**Man** Son1 +0 Son2 +5 +10 Son3 +0 Wife +0 Daughter +0 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son2** Man -5 -10 Son1 +0 Son3 +0 Wife +0 Daughter -10 **Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Man** Son1 15 Son2 15 Son3 0 Wife 0 Daughter 0 **Son1** Man -15 Son2 0 Son3 0 Wife 0 Daughter -10 **Son2** Man -15 Son1 0 Son3 0 Wife 0 Daughter -10 **Son3** Man 0 Son1 0 Son2 0 Wife 0 Daughter 0

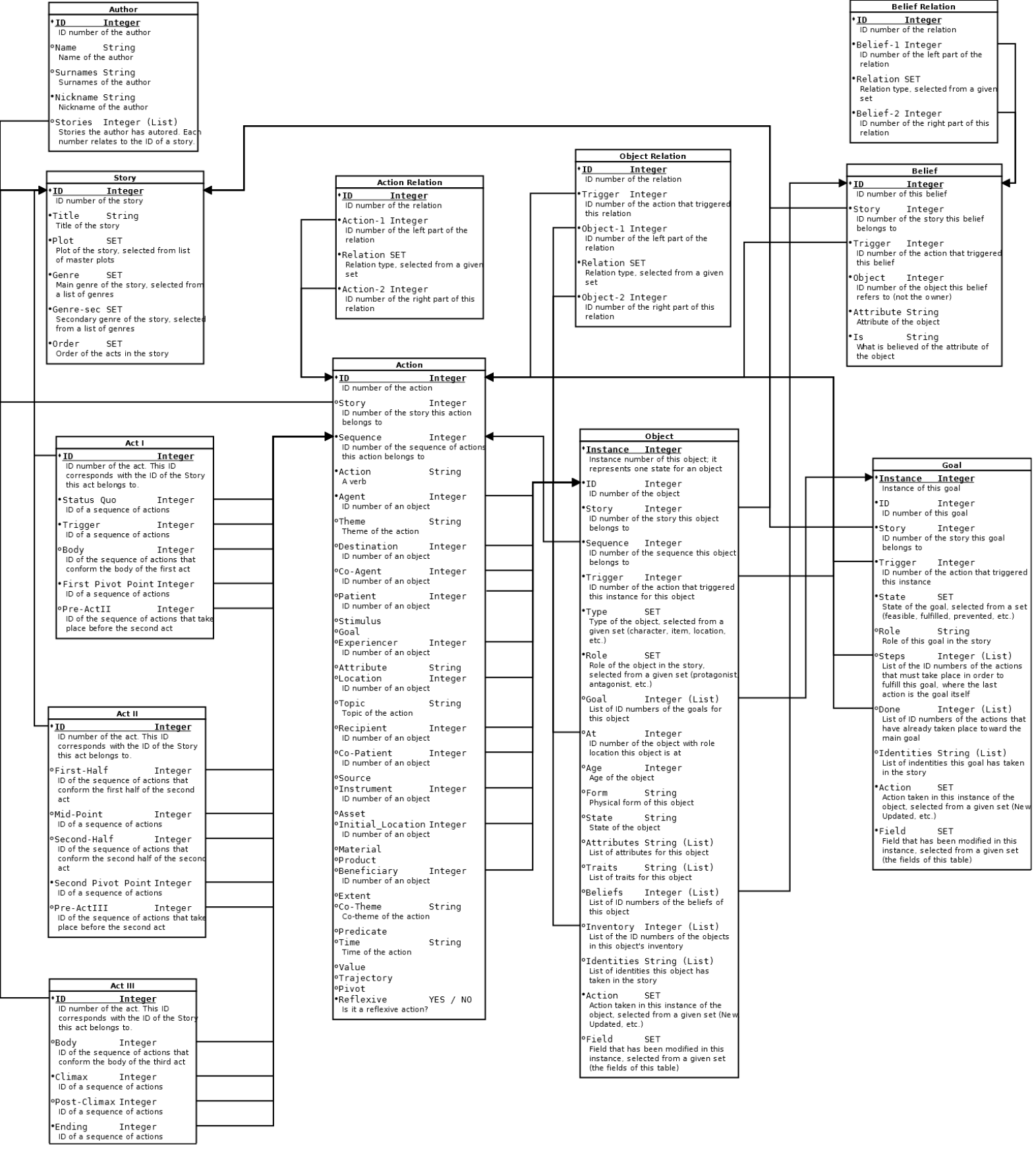
ACT II				When the youngest son was set to watch, and claimed that their sister had gone to the outhouse, he claimed that the cows must have died from seeing the moon.	Third son is set [to watch]. **Third son's goal:** Discover who kills cows at night **Third son's goal fulfilled.** Third son claims that [daughter has gone to the outhouse] and [cows die because [cows look at the moon]]. **Man's goal fulfilled.**	[Event] Set someone -> [inferes] Order [Event] Watch [Event] Claim [Event] Die [Event] Look at [Event] Believe	**[U] Son3:** **Role**: Antagonist + **Goal**: GoalSon3 **Atributes:** First **Relationships:** [Son] Man [Brother] Son1 [Brother] Son2 [Brother] Daughter	**+ GoalSon3:** **State:** Feasible **Steps:** watch(cows) discover(who(kill(cows)), night)	**[U] GoalSon3:** **State:** Feasible **Steps:** discover(who(kill(cows)), night) **[U]Done:** + watch(cows)	**[U] GoalSon3:** **State:** Feasible **Steps:** **[U]Dons:** watch(cows) + discover(who(kill(cows)), night)	**[U] GoalSon3:** **[U]State:** Fulfilled **Steps:** **Done:** watch(cows) discover(who(kill(cows)), night)	**[U] GoalMan:** **[U]State:** Feasible **Steps:** discover(who(kill(cows)), night)		Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Son3 | Learn Curse = 10 | Daughter Son3 | Lie = 5 | Man	**Man** Son1 +0 Son2 +0 Son3 -5 Wife +0 Daughter +0 **Son3** Man +5 Son1 +0 Son2 +0 Wife +0 Daughter -10 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Wife** Son1 +0 Son2 +0 Son3 +0 Man +0 Daughter +0 **Son2** Man+0 Son1 +0 Son3 +0 Wife +0 Daughter +0 **Daughter** Son1 +0 Son2 +0 Son3 +10 Man +0 Wife +0	**Man** Son1 15 Son2 15 Son3 -5 Wife 0 Daughter 0 **Son3** Man 5 Son1 0 Son2 0 Wife 0 Daughter -10 **Son1** Man -15 Son2 0 Son3 0 Wife 0 Daughter -10 **Wife** Son1 0 Son2 0 Son3 0 Man 0 Daughter 0 **Son2** Man -15 Son1 0 Son3 0 Wife 0 Daughter -10 **Daughter** Son1 10 Son2 10 Son3 10 Man 0 Wife 0
	MID-POINT			The older brothers wandered until they met a Buddhist monk, who sent them back with three magical bottles.	First and second son wander. First son and second son meet a Buddhist monk. Buddhist monk is secondary. **Buddhist monk's goal:** To help first son. **Buddhist monk's goal:** To help second son. Buddhist monk gives First son and Second son a white bottle, a blue bottle and a red bottle. White bottle, blue bottle and red bottle are magical. Buddhist monk sends First son and Second son back.	[Event] Wander [Event] Meet -> [inferes] [Relationship] Acquaintance [Person] Monk [Attribute] Buddhist [Event] Give [Item] Bottle [Attribute] White [Attribute] Blue [Attribute] Red [Attribute] Magical [Event] Send someone back -> [inferes] Order	**+ WhiteBottle:** **Role:** Item **Object:** Bottle **Atributes:** White Magical **+ BlueBottle:** **Role:** Item **Object:** Bottle **Atributes:** Blue Magical **+ RedBottle:** **Role:** Item **Object:** Bottle **Atributes:** Red Magical	**+ Monk:** **Role:** Secondary **Goal:** Goal1Monk Goal2Monk **Atributes:** Buddhist **Relationships:** [Acquaintance] Son1 [Acquaintance] Son2	**[U] Son1:** **Role**: Antagonist **Atributes:** First + **Inventory:** + WhiteBottle + BlueBottle + RedBottle **[U]Relationships:** [Son] Man [Brother] Son2 [Brother] Son3 [Brother] Daughter + [Acquaintance] Monk	**[U] Son2:** **Role**: Antagonist **Atributes:** First + **Inventory:** + WhiteBottle + BlueBottle + RedBottle **Relationships:** [Son] Man [Brother] Son1 [Brother] Son3 [Brother] Daughter + [Acquaintance] Monk	**+ Goal1Monk:** **State:** Feasible **Steps:** help(Son1) **+ Goal2Monk:** **State:** Feasible **Steps:** help(Son2)		Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) + Monk (S)	**Man** Son1 +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 Monk +0 **Wife** Son1 +0 Son2 +0 Son3 +0 Man +0 Daughter +0 **Son2** Man+0 Son1 +0 Son3 +0 Wife +0 Daughter +0 Monk +0 **Daughter** Son1 +0 Son2 +0 Son3 +0 Man +0 Wife +0	**Man** Son1 15 Son2 15 Son3 -5 Wife 0 Daughter 0 **Son3** Man 5 Son1 0 Son2 0 Wife 0 Daughter -10 **Son1** Man -15 Son2 0 Son3 0 Wife 0 Daughter -10 Monk 0 **Wife** Son1 0 Son2 0 Son3 0 Man 0 Daughter 0 **Son2** Man -15 Son1 0 Son3 0 Wife 0 Daughter -10 Monk 0 **Daughter** Son1 10 Son2 10 Son3 10 Man 0 Wife 0
		Evolving relationships between the antagonist and the metamorph. The characters will generally move toward each other emotionally.	Event involving Protagonist and Antagonist	They found their sister living alone;	First son and Second son find daughter living alone.	[Event] Find [Event] Living (alone)						Event involving Protagonist and Antagonist P [E] A Or A [E] P ??? **Yes** Daughter [persuades] Son1 Daughter [persuades] Son2	Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Monk (S)	**Man** Son1 +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 Monk +0 **Wife** Son1 +0 Son2 +0 Son3 +0 Man +0 Daughter +0 **Son2** Man+0 Son1 +0 Son3 +0 Wife +0 Daughter +0 Monk +0 **Daughter** Son1 +0 Son2 +0 Son3 +0 Man +0 Wife +0	**Man** Son1 15 Son2 15 Son3 -5 Wife 0 Daughter 0 **Son3** Man 5 Son1 0 Son2 0 Wife 0 Daughter -10 **Son1** Man -15 Son2 0 Son3 0 Wife 0 Daughter -10 Monk 0 **Wife** Son1 0 Son2 0 Son3 0 Man 0 Daughter 0 **Son2** Man -15 Son1 0 Son3 0 Wife 0 Daughter -10 Monk 0 **Daughter** Son1 10 Son2 10 Son3 10 Man 0 Wife 0
				she told them their parents and brother had died, and implored them to stay.	Daughter tells First son and Second son that [man, man's wife and third son have died].	[Event] Tell [Event] Die -> [inferes] [State] Dead	**[U] Man:** **Role**: Secondary + **State:** Dead **Relationships:** [Father] Son1 [Father] Son2 [Father] Son3 [Husband] Wife [Father] Daughter	**[U] Wife:** **Role**: Secondary + **State:** Dead **Relationships:** [Wife] Man [Mother] Daughter	**[U] Son3:** **Role**: Antagonist + **State:** Dead **Atributes:** First **Relationships:** [Son] Man [Brother] Son1 [Brother] Son2 [Brother] Daughter				Man (S) Son1 (A) Son2 (A) Son3 (A) Wife(S) Daughter(P) Monk (S)	**Man** Son1 +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 **Son3** Man +0 Son1 +0 Son2 +0 Wife +0 Daughter +0 **Son1** Man +0 Son2 +0 Son3 +0 Wife +0 Daughter +0 Monk +0 **Wife** Son1 +0 Son2 +0 Son3 +0 Man +0 Daughter +0 **Son2** Man+0 Son1 +0 Son3 +0 Wife +0 Daughter +0 Monk +0 **Daughter** Son1 +0 Son2 +0 Son3 +0 Man +0 Wife +0	**Man** Son1 15 Son2 15 Son3 -5 Wife 0 Daughter 0 **Son3** Man 5 Son1 0 Son2 0 Wife 0 Daughter -10 **Son1** Man -15 Son2 0 Son3 0 Wife 0 Daughter -10 Monk 0 **Wife** Son1 0 Son2 0 Son3 0 Man 0 Daughter 0 **Son2** Man -15 Son1 0 Son3 0 Wife 0 Daughter -10 Monk 0 **Daughter** Son1 10 Son2 10 Son3 10 Man 0 Wife 0
	SECOND PIVOT/ POINT			Finally, she persuaded them to stay the night and somehow made a rich meal for them.	Daughter persuades first son and second son to stay the night. Daughter makes rich meal for First son and Second son.	[Event] Persuade [Event] Stay [Time] Night [Item] Meal [Attribute] Rich	**+ Meal:** **Role:** Item **Object:** Bottle **Atributes:** Rich								
ACT III				In the night, the older brother was woken by the sounds of chewing.	It is night. First son wakes by sounds of chewing.	[Time] Night [Event] Wake									
				He rolled over, saw the meal, and realized that they had been eating corpses.	First son rolls over. First son sees the meal. First son realizes the meal was corpses.	[Event] Roll over [Event] See [Item] Corpse	**[U] Meal:** **Role:** Item **[U]Object:** Corpse **Atributes:** Rich								
				The sister stood over his dead brother, eating his liver.	Daughter stands over second son. Second son is dead. Daughter is eating second son's liver.	[Event] Stand over [State] Dead [Body] Liver	**[U] Son2:** **Role**: Antagonist + **State:** Dead **Atributes:** First **Inventory:** WhiteBottle BlueBottle RedBottle **Relationships:** [Son] Man [Brother] Son1 [Brother] Son3 [Brother] Daughter [Acquaintance] Monk	**+ Goal2Monk:** **State:** Unfeasible **Steps:** help(Son2)							
				She told him that she needed only one more to become a human.	Daughter tells first son that [daughter needs one more liver [to become human]]. **Update Daughter's goal:** **Role:** Curse release Step(1): eat(liver(1))	[Event] Tell	**[U] GoalProtagonist:** **State:** Feasible **Role:** Curse + **Steps:** eat(liver(1)) become(human) **Other identities:** Curse								
	CLIMAX	The terms of the release are fulfilled by the antagonist and the protagonist is freed from the curse.	The antagonist fulfills the release for the curse.	He fled, throwing the white bottle behind him, and it became a thicket of thorns.	**First son's goal:** To escape daughter. First son runs away First son throws the white bottle behind him. The white bottle becomes a thicket of thorns.	[Event] Run away [Event] Throw [Event] Become [Item] Thicket of thorns	**[U] WhiteBottle:** **[U]Role**: Location **[U]Form:** Thicket of thorns **Atributes:** White Magical	**[U] Son1:** **Role**: Antagonist + **Goal:** Goal2Son1 **Atributes:** First **[U]Inventory:** - WhiteBottle BlueBottle RedBottle **Relationships:** [Son] Man [Brother] Son2 [Brother] Son3 [Brother] Daughter [Acquaintance] Monk	**+ Goal2Son1:** **State:** Feasible **Steps:** escape(Daughter)		The antagonist fulfills the release for the curse. A [Event] -> P [State] +Released ??? **Yes** Death -> Daughter [State] + Released Fire [Kill] Daughter RedBottle [turn] Fire Son1 [throw] RedBottle So Son1 [throw] RedBottle [turn] Fire [kill] Daughter -> Daughter [State] + Released			
				As a fox, she made her way through it.	Daughter takes fox form. Daughter makes her way through the thicket of thorns. Daughter takes human form.	[Event] Take something form -> [inferes] become *something* -> [inferes] [State] fox [Item] Thicket (of thorns) [Event] Take something form -> [inferes] become *something* -> [inferes] [State] human	**[U] Daughter:** **Role**: Protagonist **Age:** 6 + **Form:** Fox + **Location:** WhiteBottle **Goal:** GoalProtagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3	**[U] Daughter:** **Role**: Protagonist **Age:** 6 **[U]Form:** Human - **Location:** WhiteBottle **Goal:** GoalProtagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3						
				He threw the blue bottle behind him, and trapped her in a river, but as a fox, she swam ashore.	First son throws the blue bottle behind him. The blue bottle becomes a river. Daughter takes fox form. Daughter swims to the river's shore. Daughter takes human form.	[Event] Throw [Event] Become [Event] Take something form -> [inferes] become *something* -> [inferes] [State] fox [Item] River [Event] Take something form -> [inferes] become *something* -> [inferes] [State] human	**[U] BlueBottle:** **[U]Role**: Location **[U]Form:** River **Atributes:** Blue Magical	**[U] Son1:** **Role**: Antagonist **Goal:** Goal2Son1 **Atributes:** First **[U]Inventory:** - BlueBottle RedBottle **Relationships:** [Son] Man [Brother] Son2 [Brother] Son3 [Brother] Daughter [Acquaintance] Monk	**[U] Daughter:** **Role**: Protagonist **Age:** 6 **[U]Form:** Fox + **Location:** BlueBottle **Goal:** GoalProtagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3	**[U] Daughter:** **Role**: Protagonist **Age:** 6 **[U]Form:** Human - **Location:** BlueBottle **Goal:** GoalProtagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3						

				He threw the red bottle behind, and she was trapped in fire.	First son throws the red bottle behind him. The red bottle becomes a fire. Daughter is trapped in fire. Fire burns daughter to death. Daughter is released by death. **Daughter's goal fulfilled.** **First son's goal fulfilled.** **Buddhist monk's goal fulfilled.**	[Event] Throw [Event] Become [Event] Take something form -> [inferes] become something -> [inferes] [State] fox [Item] Thicket (of thorns) [Event] Take something form -> [inferes] become something -> [inferes] [State] human [Event] Burn to death -> [inferes] [Event] Kill -> [inferes] [State] Dead [State] Released	**[U] RedBottle:** **[U]Role:** Location **[U]Form:** Fire **Atributes:** Red Magical	**[U] Son1:** **Role**: Antagonist **Goal:** Goal2Son1 **Atributes:** First **[U]Inventory:** - RedBottle **Relationships:** [Son] Man [Brother] Son2 [Brother] Son3 [Brother] Daughter [Acquaintance] Monk	**[U] Daughter:** **Role**: Protagonist **Age:** 6 **Form:** Human **+ State:** + Dead - Released **+ Location:** RedBottle **Goal:** GoalProtagonist **Relationships:** [Daughter] Man [Daughter] Wife [Sister] Son1 [Sister] Son2 [Sister] Son3	**+ Goal1Monk:** **State:** Fulfilled **Steps:** help(Son1)	**[U] GoalProtagonist:** **[U]State:** Fulfilled **Role:** Curse **Steps:** eat(liver(1)) become(human) **Other identities:** Curse
	ENDING	The metamorph may either revert to the original state or die.		It burned her until she was no more than a mosquito.	From the ashes, a mosquito is born. Mosquito is secondary	[Location] Ashes [Event] Born [Insect] Mosquito	**+ Ashes:** **Role**: Location	**+ Mosquito:** **Role**: Secondary **Location:** Ashes			

Anexo C: Esquema preliminar de StoryDB

Este documento fue originalmente exportado a una imagen PNG y ha sido impreso en formato PDF para facilitar la lectura de esta memoria.

La imagen original está disponible para cualquiera que la solicite.


Author
•ID Integer
ID number of the author
°Name String
Name of the author
°Surnames String
Surnames of the author
•Nickname String
Nickname of the author
°Stories Integer (List)
Stories the author has autored. Each number relates to the ID of a story.
Belief Relation
•ID Integer
ID number of the relation
•Belief-1 Integer
ID number of the left part of the relation
•Relation SET
Relation type, selected from a given set
•Belief-2 Integer
ID number of the right part of this relation
Story
•ID Integer
ID number of the story
•Title String
Title of the story
•Plot SET
Plot of the story, selected from list of master plots
•Genre SET
Main genre of the story, selected from a list of genres
•Genre-sec SET
Secondary genre of the story, selected from a list of genres
•Order SET
Order of the acts in the story
Action Relation
•ID Integer
ID number of the relation
•Action-1 Integer
ID number of the left part of the relation
•Relation SET
Relation type, selected from a given set
•Action-2 Integer
ID number of the right part of this relation
Object Relation
•ID Integer
ID number of the relation
•Trigger Integer
ID number of the action that triggered this relation
•Object-1 Integer
ID number of the left part of the relation
•Relation SET
Relation type, selected from a given set
•Object-2 Integer
ID number of the right part of this relation
Belief
•ID Integer
ID number of this belief
•Story Integer
ID number of the story this belief belongs to
•Trigger Integer
ID number of the action that triggered this belief
•Object Integer
ID number of the object this belief refers to (not the owner)
•Attribute String
Attribute of the object
•Is String
What is believed of the attribute of the object
Action
•ID Integer
ID number of the action
°Story Integer
ID number of the story this action belongs to
•Sequence Integer
ID number of the sequence of actions this action belongs to
•Action String
A verb
•Agent Integer
ID number of an object
°Theme String
Theme of the action
°Destination Integer
ID number of an object
°Co-Agent Integer
ID number of an object
°Patient Integer
ID number of an object
°Stimulus
°Goal
°Experiencer Integer
ID number of an object
°Attribute String
°Location Integer
ID number of an object
°Topic String
Topic of the action
°Recipient Integer
ID number of an object
°Co-Patient Integer
ID number of an object
°Source
°Instrument Integer
ID number of an object
°Asset
°Initial_Location Integer
ID number of an object
°Material
°Product
°Beneficiary Integer
ID number of an object
°Extent
°Co-Theme String
Co-theme of the action
°Predicate
°Time String
Time of the action
°Value
°Trajectory
°Pivot
•Reflexive YES / NO
Is it a reflexive action?
Act I
•ID Integer
ID number of the act. This ID corresponds with the ID of the Story this act belongs to.
•Status Quo Integer
ID of a sequence of actions
•Trigger Integer
ID of a sequence of actions
°Body Integer
ID of the sequence of actions that conform the body of the first act
•First Pivot Point Integer
ID of a sequence of actions
°Pre-ActII Integer
ID of the sequence of actions that take place before the second act
Act II
•ID Integer
ID number of the act. This ID corresponds with the ID of the Story this act belongs to.
°First-Half Integer
ID of the sequence of actions that conform the first half of the second act
°Mid-Point Integer
ID of a sequence of actions
°Second-Half Integer
ID of the sequence of actions that conform the second half of the second act
•Second Pivot Point Integer
ID of a sequence of actions
°Pre-ActIII Integer
ID of the sequence of actions that take place before the second act
Act III
•ID Integer
ID number of the act. This ID corresponds with the ID of the Story this act belongs to.
°Body Integer
ID of the sequence of actions that conform the body of the third act
•Climax Integer
ID of a sequence of actions
°Post-Climax Integer
ID of a sequence of actions
•Ending Integer
ID of a sequence of actions
Object
•Instance Integer
Instance number of this object; it represents one state for an object
•ID Integer
ID number of the object
•Story Integer
ID number of the story this object belongs to
•Sequence Integer
ID number of the sequence this object belongs to
•Trigger Integer
ID number of the action that triggered this instance for this object
•Type SET
Type of the object, selected from a given set (character, item, location, etc.)
•Role SET
Role of the object in the story, selected from a given set (protagonist, antagonist, etc.)
°Goal Integer (List)
List of ID numbers of the goals for this object
°At Integer
ID number of the object with role location this object is at
°Age Integer
Age of the object
°Form String
Physical form of this object
°State String
State of the object
°Attributes String (List)
List of attributes for this object
°Traits String (List)
List of traits for this object
°Beliefs Integer (List)
List of ID numbers of the beliefs of this object
°Inventory Integer (List)
List of the ID numbers of the objects in this object's inventory
°Identities String (List)
List of identities this object has taken in the story
•Action SET
Action taken in this instance of the object, selected from a given set (New, Updated, etc.)
°Field SET
Field that has been modified in this instance, selected from a given set (the fields of this table)
Goal
•Instance Integer
Instance of this goal
•ID Integer
ID number of this goal
•Story Integer
ID number of the story this goal belongs to
•Trigger Integer
ID number of the action that triggered this instance
•State SET
State of the goal, selected from a set (feasible, fulfilled, prevented, etc.)
°Role String
Role of this goal in the story
°Steps Integer (List)
List of the ID numbers of the actions that must take place in order to fulfill this goal, where the last action is the goal itself
°Done Integer (List)
List of ID numbers of the actions that have already taken place toward the main goal
°Identities String (List)
List of indentities this goal has taken in the story
•Action SET
Action taken in this instance of the object, selected from a given set (New, Updated, etc.)
•Field SET
Field that has been modified in this instance, selected from a given set (the fields of this table)

Anexo D: Documentación de GluNet

GluNet

Documentation

Ben Kybartas

February 9, 2016

Contents

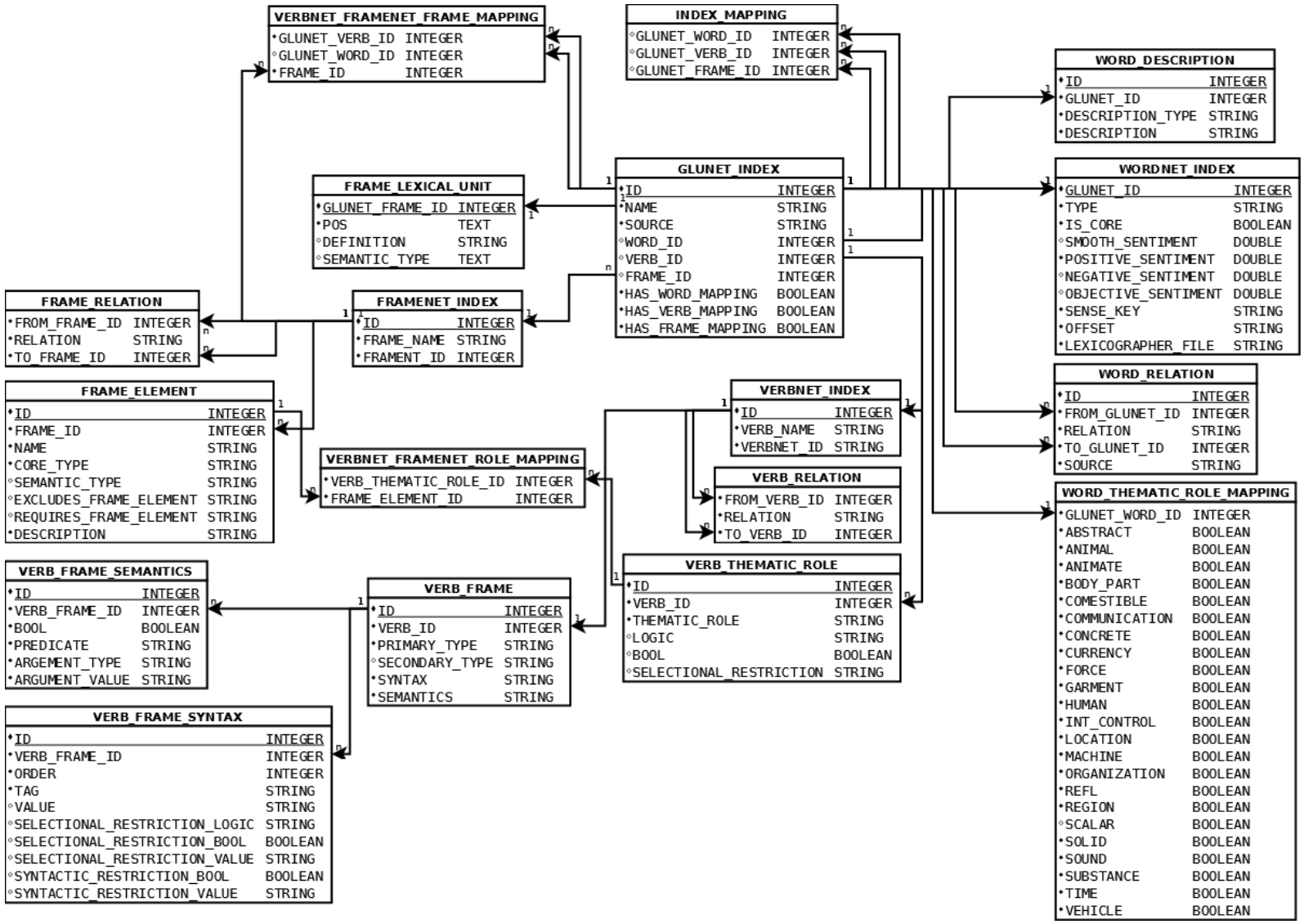


Figure 1: Database structure of GluNet.

1 Introduction

GluNet is a lexical and commonsense database. It is made with the unification of the data from WordNet [4], VerbNet [6], FrameNet [1], ConceptNet [9] and several additional sources. For a complete introduction to GluNet, please see the publication from Kybartas and Bidarra [7]. This document serves as a technical description of the actual GluNet database, as shown in Figure 1.

The GluNet database is an SQLite3 database, and it should be noted that all **BOOLEAN** values are actually stored as integers where 1 is equal to true and 0 to false.

The main index of the database is the Glunet_Index table which is composed of the unique word sense from WordNet, the verb members of VerbNet and the lexical units of FrameNet. The remainder of this document explains each of the tables in more detail, split between into categories based upon WordNet/ConceptNet data, VerbNet data, FrameNet data and the mappings.

1.1 GLUNET_INDEX

The main index of the GluNet database. The data in this database can link an individual word to all other semantics in the database.

1.1.1 Column Descriptions

- **ID**: The primary key for the database, this is the main ID for all of the GluNet database
- **NAME**: The actual word we are referring to
- **SOURCE**: The source of the word, currently one of three possible values, either WORDNET, VERBNET or FRAMENET
- **WORD_ID**: If the source is WORDNET, then the ID of this word in the WORDNET_INDEX table is given here.
- **VERB_ID**: If the source is VERBNET, then the ID of the verb in the VERBNET_INDEX table is given here.
- **FRAME_ID**: If the source is FRAMENET, then the ID of the frame in the FRAMENET_INDEX table is given here.
- **HAS_WORD_MAPPING**: True if this word is mapped to a WordNet word, and false otherwise, this mapping is stored in the INDEX_MAPPING table.
- **HAS_VERB_MAPPING**: True if this word is mapped to a VerbNet word, and false otherwise, this mapping is stored in the INDEX_MAPPING table.
- **HAS_FRAME_MAPPING**: True if this word is mapped to a FrameNet word, and false otherwise, this mapping is stored in the INDEX_MAPPING table.

2 WordNet and ConceptNet Data

Below we discuss all the tables with data taken from WordNet and ConceptNet.

2.1 WORDNET_INDEX

Data taken from WordNet [4], [3], [2], and [8]. These are semantics relating to the words themselves and not the relations between words.

2.1.1 Column Descriptions

- **GLUNET_ID**: The main ID from the GLUNET_INDEX table.
- **TYPE**: The type of the word, can be a 'noun', 'verb', 'adjective' or 'adverb'
- **IS_CORE**: A boolean which is true if the word is one of the 5000 most common words. Taken from [2].
- **SMOOTH_SENTIMENT**: A smoothed value of a word's sentiment, with larger negative values equating to a more negative sentiment of the word and vice-versa. Taken from [8].
- **POSITIVE_SENTIMENT**: A value between zero and one stating the likely hood that this word is used to express positive sentiments. Taken from [3].
- **NEGATIVE_SENTIMENT**: A value between zero and one stating the likely hood that this word is used to express negative sentiments. Taken from [3].
- **OBJECTIVE_SENTIMENT**: A value between zero and one stating the likely hood that this word is used to express objective sentiments. Taken from [3].
- **SENSE_KEY**: The sense key is part of the index used by WordNet. It will likely be of little use except for database maintenance.
- **OFFSET**: The offset is part of the index used by WordNet. It will likely be of little use except for database maintenance.
- **LEXICOGRAPHER_FILE**: The offset is part of the index used by WordNet. It will likely be of little use except for database maintenance.

2.2 WORD_DESCRIPTION

Data taken from WordNet [4]. Includes descriptions, examples and verb frames of individual words.

2.2.1 Column Descriptions

- **ID**: The primary key for the database
- **GLUNET_ID**: The main ID from the GLUNET_INDEX table.
- **DESCRIPTION_TYPE**: The type of description, can be either a literal DESCRIPTION of the word, an EXAMPLE of the word in a sentence, or a VERB_FRAME of the word showing its syntax. Taken from [4]
- **DESCRIPTION**: The actual text containing the description, example or verb frame.

2.3 WORD_RELATION

Data taken from WordNet [4], [2], [5] and ConceptNet [9]. Contains all relations between two words. This is the largest database in GluNet. All relations should be assumed to be one way.

2.3.1 Column Descriptions

- **ID**: The primary key for the database
- **FROM_GLUNET_ID**: The main ID from the GLUNET_INDEX table, representing the word from which the relationship exists.
- **RELATION**: The type of relation between words. These relations come from all four of the above-mentioned references and there are a total of 63 possible relations, such as SYNONYM, PARENT_OF, AT_LOCATION etc.
- **TO_GLUNET_ID**:The main ID from the GLUNET_INDEX table, representing the word to which the relationship applies.
- **SOURCE**: The source of the relation, can be either WORDNET, for data taken from [4], MORPHOSEMANTIC for data taken from [5], TELEOLOGICAL for data taken from [2], or CONCEPTNET for data taken from [9]

3 Verbnet Data

Below we discuss all the tables with data taken from VerbNet. All data in this section is solely from VerbNet [6]

3.1 VERBNET_INDEX

The main index of VerbNet, each verb member from GLUNET_INDEX is linked to one of the main verb classes in this table as multiple verb members can have the same semantics.

3.1.1 Column Descriptions

- **ID**: The primary key of the table
- **VERB_NAME**: The name of the verb class
- **VERBNET_ID**: The index used by VerbNet. It will likely be of little use except for database maintenance.

3.2 VERB_RELATION

The relations between different verb classes. Currently this only includes parent/child relations.

3.2.1 Column Descriptions

- **FROM_VERB_ID**: The ID of the verb class, from VERBNET_INDEX, which is the origin of the relation.
- **RELATION**: The relation between verb classes, currently there is only one PARENT, indicating a hierarchical relation from one verb to another.
- **TO_VERB_ID**: The ID of the verb class, from VERBNET_INDEX, which is the destination of the relation.

3.3 VERB_THEMATIC_ROLE

The thematic roles associated with a verb. Note that each selectional restriction gets its own entry, meaning that some thematic roles will be listed twice.

3.3.1 Column Descriptions

- **ID**: The primary key of the table
- **VERB_ID**: The ID of the verb class, from VERBNET_INDEX
- **THEMATIC_ROLE**: The thematic role associated with this verb class. May be one of 30 possible values, such as Agent, Theme, Destination, etc.
- **LOGIC**: If there is a selectional restriction with some form of logic, either 'and' or 'or', it is listed here.
- **BOOL**: The boolean which states whether the selectional restriction is true (the thematic role must be this) or false (the thematic role cannot be this).
- **SELECTIONAL_RESTRICTION**: the actual selectional restriction. May be one of 27 possible values, such as animate, organization, communication, etc. See WORD_THEMATIC_ROLE_MAPPING for a mapping of which WordNet words may fulfill these selectional restrictions (although note that the mapping is incomplete and does not cover all thematic roles).

3.4 VERB_FRAME

Each verb class has one or more frames, each with their own syntax and semantics (both of which get tables of their own). This class contains the verb frame semantics and summary descriptions of the syntax and semantics.

3.4.1 Column Descriptions

- **ID**: The primary key of the table.
- **VERB_ID**: The ID of the verb class, from VERBNET_INDEX.
- **PRIMARY_TYPE**: The primary type of the verb frame. May be one of 288 possible values, such as NP V NP, NP V how S, NP V S_ING, etc.
- **SECONDARY_TYPE**: The optional secondary type of the verb frame. May be one of 400 possible values, such as HOW-S, Basic Transitive, POSSING, etc.
- **SYNTAX**: The full syntax. For a detailed version of this syntax see VERB_FRAME_SYNTAX.
- **SEMANTICS**: The full semantics. For a detailed version of this syntax see VERB_FRAME_SEMANTICS.

3.5 VERB_FRAME_SEMANTICS

A detailed version of the semantics for each verb frame, predicates may be listed more than once as each predicate argument gets its own entry.

3.5.1 Column Descriptions

- **ID**: The primary key of the table.
- **VERB_FRAME_ID**: The ID of the verb frame, from VERB_FRAME.
- **BOOL**: 1 if the predicate should be taken as true, 0 otherwise. For example 1 for the predicate 'approve' indicates approval, while 0 would indicate '!approve' or disapproval.
- **PREDICATE**: the predicate, may be one of 146 possible values, such as approve, motion, yield, etc.
- **ARGUMENT_TYPE**: The type of the argument used for the predicate. May be either an Event, ThemRole (thematic role), VerbSpecific or Constant.
- **ARGUMENT_VALUE**: The value of the argument for the predicate. May be one of 97 possible values, such as Agent, Theme, during(E), etc. (Note that E always refers to the Event itself).

3.6 VERB_FRAME_SYNTAX

A detailed version of the syntax for each verb frame. Each selectional restriction gets its own entry meaning some tags may appear more than once.

3.6.1 Column Descriptions

- **ID**: The primary key of the table
- **VERB_FRAME_ID**: The ID of the verb frame, from VERB_FRAME.
- **ORDER**: Since ordering matters, this value indicates the position of the tag in the sentence. 0 means start of the sentence, 1 means second position in the sentence, etc.
- **TAG**: The tag at that particular position in the sentence, may be NP, VERB, PREP, ADV, LEX or ADJ
- **VALUE**: The optional value of each tag, can be one of 94 possible values, such as Agent, Theme, Destination, etc.
- **SELECTIONAL_RESTRICTION_LOGIC**: If the tag has a selectional restriction, and that restriction has some logic (either 'and' or 'or'), then it is listed here.
- **SELECTIONAL_RESTRICTION_BOOL**: 1 if the restriction must be met, 0 if it must be avoided.
- **SELECTIONAL_RESTRICTION_VALUE**: The actual value of the restriction. May be one of 13 possible values such as location, region, state, etc.
- **SYNTACTIC_RESTRICTION_BOOL**: If the tag has a syntactic restriction, 1 if the syntactic restriction needs to be met, and 0 if it needs to be avoided.
- **SYNTACTIC_RESTRICTION_VALUE**: The value of the syntactic restriction, may be one of 40 possible values, such as how_extract, sentential, poss_ing, etc.

4 FrameNet Data

Below we discuss all the tables with data taken from FrameNet. All data in this section is solely from FrameNet [1].

4.1 FRAMENET_INDEX

The index of FrameNet. Each lexical unit in the GLUNET_INDEX is linked to one of these frames.

4.1.1 Column Descriptions

- **ID**: The primary key of the table.
- **FRAME_NAME**: The name of the frame.
- **FRAMENET_ID**: The index used by FrameNet. It will likely be of little use except for database maintenance.

4.2 FRAME_LEXICAL_UNIT

The data for each lexical unit in the GLUNET_INDEX.

4.2.1 Column Descriptions

- **GLUNET_FRAME_ID**: The ID from the GLUNET_INDEX
- **POS**: The type of the lexical unit, can be one of 10 possible values, such as V (verb), N (noun), PREP, etc.
- **DEFINITION**: The definition of the lexical unit.
- **SEMANTIC_TYPE**: The optional semantic type of the unit. May be one of 34 possible values, such as Landform, Container, Support, etc.

4.3 FRAME_RELATION

The relations between frames in FrameNet.

4.3.1 Column Descriptions

- **FROM_FRAME_ID**: The ID of the origin frame, from FRAMENET_INDEX.
- **RELATION**: The type of relation existing between frames, may be one of 8 possible values, CAUSATIVE_OF, INCHOATIVE_OF, INHERITS_FROM, PERSPECTIVE_ON, PRECEDES, SEE_ALSO, SUBFRAME or USING.
- **TO_FRAME_ID**: The ID of the destination frame, from FRAMENET_INDEX.

4.4 FRAME_ELEMENT

The different elements of each FrameNet frame.

4.4.1 Column Descriptions

- **ID**: The primary key of the table
- **FRAME_ID**: The ID of the frame, from FRAMENET_INDEX
- **NAME**: The name of the frame element, may be one of 1170 possible values.
- **CORE_TYPE**: The core type of the frame element, may be Core, Peripheral, Extra-Thematic, Core-Unexpressed.
- **SEMANTIC_TYPE**: The optional semantic type of the frame element, may be one of 31 possible values, such as Physical_Object, Location, Degree, etc.
- **EXCLUDES_FRAME_ELEMENT**: If the frame element excludes another frame element, it is listed here.
- **REQUIRES_FRAME_ELEMENT**: If the frame element requires another frame element, it is listed here.
- **DESCRIPTION**: A description of the frame element.

5 Mappings

Below we discuss all the tables dealing with the mappings between WordNet, VerbNet and FrameNet. Data is taken from the SemLink project associated with VerbNet [6], the original VerbNet/WordNet mappings from VerbNet [6], and [10]

5.1 INDEX_MAPPING

The main index mapping between the GLUNET_INDEX IDs, relating to matchings between WordNet words, VerbNet verb members and FrameNet lexical units. This mapping is a combination of the above mentioned sources.

5.1.1 Column Descriptions

- **GLUNET_WORD_ID**: The WordNet word ID, taken from GLUNET_INDEX.
- **GLUNET_VERB_ID**: The VerbNet verb member ID, taken from GLUNET_INDEX.
- **GLUNET_FRAME_ID**: The FrameNet frame lexical unit ID, taken from GLUNET_INDEX.

5.2 VERBNET_FRAMENET_FRAME_MAPPING

The mapping between VerbNet verb members and actual frames (as opposed to simple the lexical units of the frame). This data is taken from the SemLink project.

5.2.1 Column Descriptions

- **GLUNET_VERB_ID**: The VerbNet verb member ID, taken from GLUNET_INDEX.
- **GLUNET_WORD_ID**: If the verb is mapped to a WordNet word (in the INDEX_MAPPING), then this mapping is also included here.
- **FRAME_ID**: The FrameNet frame ID, taken from FRAMENET_INDEX.

5.3 VERBNET_FRAMENET_ROLE_MAPPING

The mapping between the thematic roles of the VerbNet verbs, and the FrameNet frame elements. This data is taken from the SemLink project.

5.3.1 Column Descriptions

- **VERB_THEMATIC_ROLE_ID**: The ID of the VerbNet verb thematic role, taken from VERB_THEMATIC_ROLE.
- **FRAME_ELEMENT_ID**: The ID of the FrameNet frame element, taken from FRAME_ELEMENT.

5.4 WORD_THEMATIC_ROLE_MAPPING

A hand-made mapping of WordNet words to the different thematic roles from VerbNet. This listing is incomplete, and not validated. Certain thematic roles, such as 'elongated' and 'pointy' were not included, as they refer to properties of objects which are not included in any of the data used. As such this table should be used with those restrictions in mind. Future work should aim to create a similar table for the FrameNet frame elements. The maps each word from WordNet to which potential thematic roles it can fill.

5.4.1 Column Descriptions

- **GLUNET_WORD_ID**: The ID of the word, taken from GLUNET_INDEX.
- For brevity's sake, each of the remain columns in this table map directly to a thematic role from VerbNet. The column is a simple boolean where 1 means the word is a potential candidate for a given thematic role, and 0 if it is not.

References

[1] C. F. Baker, C. J. Fillmore, and J. B. Lowe. The Berkeley FrameNet project. In *COLING-ACL '98: Proceedings of the Conference*, pages 86–90, Montréal, Canada, 1998.

[2] J. Boyd-Graber, C. Fellbaum, D. Osherson, and R. Schapire. Adding dense, weighted connections to WordNet. In *In Proceedings of the Third International WordNet Conference*, 2006.

[3] A. Esuli and F. Sebastiani. SENTIWORDNET: A publicly available lexical resource for opinion mining. In *In Proceedings of the 5th Conference on Language Resources and Evaluation*, pages 417–422, 2006.

[4] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[5] C. Fellbaum, A. Osherson, and P. E. Clark. Putting semantics into WordNet's "morphosemantic" links. In Z. Vetulani and H. Uszkoreit, editors, *Human Language Technology. Challenges of the Information Society*, volume 5603 of *Lecture Notes in Computer Science*, pages 350–358. Springer Berlin Heidelberg, 2009.

[6] K. Kipper, A. Korhonen, N. Ryant, and M. Palmer. A large-scale classification of english verbs. *Language Resources and Evaluation*, 42(1):21–40, 2008.

[7] B. Kybartas and R. Bidarra. *Interactive Storytelling: 8th International Conference on Interactive Digital Storytelling, ICIDS 2015, Copenhagen, Denmark, November 30 - December 4, 2015, Proceedings*, chapter A Semantic Foundation for Mixed-Initiative Computational Storytelling, pages 162–169. Springer International Publishing, Cham, 2015.

[8] B. Li, M. Thakkar, Y. Wang, and M. Riedl. Storytelling with adjustable narrator styles and sentiments. In A. Mitchell, C. Fernández-Vara, and D. Thue, editors, *Interactive Storytelling*, volume 8832 of *Lecture Notes in Computer Science*, pages 1–12. Springer International Publishing, 2014.

[9] H. Liu and P. Singh. ConceptNet - a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226, 2004.

[10] L. Shi and R. Mihalcea. Putting pieces together: Combining FrameNet, VerbNet and WordNet for robust semantic parsing. In A. Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 3406 of *Lecture Notes in Computer Science*, pages 100–111. Springer Berlin Heidelberg, 2005.

Anexo E: Documentación de StoryDB

StoryDB

Documentation

Olatz Castaño

June 6, 2016

Contents

1. Introduction

StoryDB is designed to hold and represent stories in a comprehensible, coherent manner. It also serves the double purpose of aiding TaleBox, a mobile app that can be used to input stories to this database.

This document serves as a technical description of the actual StoryDB database.

The StoryDB database is an SQLite3 database, and it should be noted that all BOOLEAN values are actually stored as integers, where 1 is equal to true and 0 is equal to false.

It should also be noted that StoryDB's tables are divided by their functionality. *App_* named tables are used by TaleBox, where *Story_* named tables are used to hold the stories stored.

The data stored in StoryDB through TaleBox was aided by the GluNet database.

2. App_

App_ tables are used by TaleBox's server to store relevant data about users and the games taking place.

2.1. App_User

The table that holds all users in TaleBox. The data in this database is the base for TaleBox to function properly, acting as credentials to log into the system and linking users to stories, moves, hands, turns and other useful data for the server.

2.1.1. Column Descriptions

- **ID:** Identification number of the user. Primary Key to the table.
- **Username:** Word that will be used as a nickname to log in TaleBox's system. Unique for every user, case sensitive, can not be null. This will be used to link the stories to the users.
- **Name:** Name of the user. Can be null.
- **Surname:** Surname of the user. Can be null.
- **Stories:** List of story ID's that the user has started.
- **Password:** Password to be used along the username to log in TaleBox's system. Hashed using SHA-256. Can not be null.
- **PlayedTurns:** Total of played turns by the user.

- **MissedTurns:** Total of missed turns by the user.

2.2. App_Game_Notifications

This table holds all system notifications for the users in TaleBox. TaleBox's system needs to notify users about turns being missed, beginning for them, etc.. Each row represents a message.

2.2.1. Column Descriptions

- **ID:** Identification number of the notification. Primary Key to the table.
- **Username:** Username to the user the notification belong to. Can not be null.
- **Message:** System notification for the user. Can not be null.
- **Seen:** True if the user has received and seen the message. False if not. Can not be null.

2.3. App_Game_Settings

The table that holds the settings for every story being played in TaleBox.

2.3.1. Column Descriptions

- **StoryID:** Identification number of a story, linking to its description. Primary Key to the table.
- **Length:** Length of the story. Used by TaleBox's server to determine how many rounds of turn a story has. Can not be null.
- **MaxPlayers:** Maximum number of players allowed in a story. Used by TaleBox's server to determine if a story game is full or can allow users to join. Can not be null.
- **ActionsPerTurn:** Number of actions / moves that can be played in a turn. Can not be null.
- **TimeLimit:** Number of minutes that a turn lasts. Used by TaleBox's server to determine when a turn was missed. Can not be null.
- **Privacy:** Privacy setting for the game. Determines when a user can or not read a story in TaleBox.
- **Joining:** Joining requests setting for the game. Determines when a user can or not

request to join a story, or when can a player of a story invite another user to join in.

- **Validation:** Request validation setting for the game. Determines how a joining request is handled in a game.

2.4. App_Game_State

The table that holds the current state of a story being played in TaleBox. Each row holds the *last* state of the game, being updated as the story progresses.

2.4.1. Column Descriptions

- **StoryID:** Identification number of a story, linking to its description. Primary Key to the table.
- **Author:** Username to the user that started this story. Can not be null.
- **CoAuthors:** List of users that have played this story, both current and past. Can be empty, as a story can be played by only one user.
- **CurrentPlayers:** List of the players currently playing this story. Can not be null.
- **TurnQueue:** List of the players remaining to play their turns in the current round. First username belongs to the player that must fulfill the current turn. Can not be null.
- **RemainingStoryPoints:** List of all the Story Points the story is left to traverse as the game advances. First Story Point addresses the current Story Point being played. Can not be null.
- **JoinRequests:** List of the users that have requested to join this story. Can be null.
- **LastTurn_TimeStamp:** Time stamp of the beggining of a new turn. Used by TaleBox's server to calculate when a turn has been missed. Can not be null.

2.5. App_Game_Scores

The table that holds the scores for every player taking part in a story. Used by TaleBox's system to enhance gameplay, throw out inactive playes and allow for challenges.

2.5.1. Column Descriptions

- **StoryID:** Identification number of a story, linking to its description. Primary Key to the table.
- **SequenceID:** Identification number of the sequence of the story this score belongs to. Can not be null.
- **Username:** Username of the player this score belongs to. Can not be null.
- **Score**: Score of the player in this sequence for this story. Can not be null.

2.6. App_Hands

The table that holds the cards that have been dealt for the current turn of a story in TaleBox.

2.6.1. Column Descriptions

- **StoryID:** Identification number of a story, linking to its description. Primary Key to the table.
- **Cards:** List of cards dealt by TaleBox's server for the current turn. A string encoded in TaleBox's form of communication with its clients, containing information about each card and verb dealt. Can not be null.

3. Story_

Story_ tables hold full stories, along with their events, characters and objects. They are linked through IDs and recording each step of the story, allowing software to extract this information, do research on it and / or expand on it.

3.1. Story_Description

The table that holds the primary information to the all stories in StoryDB. Links every table together to represent the story.

3.1.1. Column Descriptions

- **ID:** Identification number of the story. Primary Key to the table.
- **Title:** Title of the Story.

- **Plot:** Name of the master plot of the story, usually one of the twenty master plots of Ronald Tobias.
- **Genre:** The genre this story belongs to (fantasy, science fiction, etc.).
- **Order:** Order of the its acts. As some stories may have deliberately unsorted acts, this sequence (123, for example) points to the order in which acts should be read.

3.2. Story_Act_I

The table that holds the IDs to each sequence of actions in the first act of the story. Every column stores an ID number for a sequence of actions. Every ID must be unique to each sequence and story.

3.2.1. Column Descriptions

- **ID:** Identification number of the story this act belongs to. Primary Key to the table.
- **StatusQuo:** Identification number of the StatusQuo sequence of actions.
- **Trigger:** Identification number of the Trigger sequence of actions.
- **Body:** Identification number of the sequence of actions pertaining to the body of the first act.
- **FirstPivotPoint:** Identification number of the First Pivot Point sequence of actions.
- **PreAct_II:** Identification number of the sequence of actions leading to act two.

3.3. Story_Act_II

The table that holds the IDs to each sequence of actions in the second act of the story. Every column stores an ID number for a sequence of actions. Every ID must be unique to each sequence and story.

3.3.1. Column Descriptions

- **ID:** Identification number of the story this act belongs to. Primary Key to the table.
- **FirstHalf:** Identification number of the sequence of action pertaining to the first half of the second act.

- **MidPoint:** Identification number of the Mid Point sequence of actions.
- **SecondHalf:** Identification number of the sequence of actions pertaining to the second half of the second act.
- **SecondPivotPoint:** Identification number of the StatusQuo sequence of actions.
- **PreAct_III:** Identification number of the StatusQuo sequence of actions.

3.4. Story_Act_III

The table that holds the IDs to each sequence of actions in the third act of the story. Every column stores an ID number for a sequence of actions. Every ID must be unique to each sequence and story.

3.4.1. Column Descriptions

- **ID:** Identification number of the story. Primary Key to the table.
- **Body:** Identification number of the StatusQuo sequence of actions.
- **Climax:** Identification number of the StatusQuo sequence of actions.
- **PostClimax:** Identification number of the StatusQuo sequence of actions.
- **Ending:** Identification number of the StatusQuo sequence of actions.

3.5. Story_Action

This table holds the semantic values for every action of a story. Storing semanting values instead of the sentence allows for deeper analysis of it. TaleBox is able to translate the move's syntax to its semantics values thanks to GluNet, that also enables to translate it back.

More on the nature of this semantics can be found in VerbNet's documentation (http://verbs.colorado.edu/~mpalmer/projects/verbnet.html).

3.5.1. Column Descriptions

- **ID:** Identification number of the action. Primary Key to the table.
- **StoryID:** Identification number of the story this action belongs to. Can not be null.
- **SequenceID:** Identification number of the story point this action belongs to.
- **Author:** The username of the user that started this story or the name of its author.

Can not be null.

- **VerbID:** The identification number in GluNet of the verb used in this action. Can not be null.
- **VerbFrameID:** The identification number in GluNet of the frame verb used in this action. Can not be null.
- **Agent:** The identification number of the story object that acts as an agent for this action. Can be null.
- **Asset:** The identification number of the story object that acts as an asset for this action. Can be null.
- **Attribute:** The identification number of the story object that acts as an attribute for this action. Can be null.
- **Beneficiary:** The identification number of the story object that acts as a beneficiary for this action. Can be null.
- **Cause:** The identification number of the story object that acts as a cause for this action. Can be null.
- **CoAgent:** The identification number of the story object that acts as a co-agent for this action. Can be null.
- **CoPatient:** The identification number of the story object that acts as a co-patientfor this action. Can be null.
- **CoTheme:** The identification number of the story object that acts as a co-theme for this action. Can be null.
- **Destination:** The identification number of the story object that acts as a destination for this action. Can be null.
- **Experiencer:** The identification number of the story object that acts as an experiencer for this action. Can be null.
- **Extent:** The identification number of the story object that acts as an extent for this action. Can be null.
- **Goal:** The identification number of the story object that acts as a goal for this action. Can be null.
- **InitialLocation:** The identification number of the story object that acts as an initial location for this action. Can be null.

- **Instrument:** The identification number of the story object that acts as an instrument for this action. Can be null.
- **Location:** The identification number of the story object that acts as a location for this action. Can be null.
- **Material:** The identification number of the story object that acts as a material for this action. Can be null.
- **Patient:** The identification number of the story object that acts as a patient for this action. Can be null.
- **Pivot:** The identification number of the story object that acts as a pivot for this action. Can be null.
- **Predicate:** The identification number of the story object that acts as a predicate for this action. Can be null.
- **Product:** The identification number of the story object that acts as a product for this action. Can be null.
- **Recipient:** The identification number of the story object that acts as a recipient for this action. Can be null.
- **Result:** The identification number of the story object that acts as a result for this action. Can be null.
- **Source:** The identification number of the story object that acts as a source for this action. Can be null.
- **Stimulus:** The identification number of the story object that acts as a stimulus for this action. Can be null.
- **Theme:** The identification number of the story object that acts as a theme for this action. Can be null.
- **Time:** The identification number of the story object that acts as a timefor this action. Can be null.
- **Topic:** The identification number of the story object that acts as a topic for this action. Can be null.
- **Trajectory:** The identification number of the story object that acts as a trajectory for this action. Can be null.
- **Value:** The identification number of the story object that acts as a value for this

action. Can be null.

- **Reflexive:** An integer that determines whether this action is reflexive or not. Can not be null.

3.6. Story_Belief

This table holds all the beliefs happening on a story. They are created through actions in the story. Beliefs hold values to an object's attribute. Beliefs are owned by objects and represent what they *think* about an attribute of the object the belief points to, that is, the value the owner gives to an attribute of the object being pointed to.

3.6.1. Column Descriptions

- **ID:** Identification number of the story. Primary Key to the table.
- **StoryID:** Identification number of the story this belief belongs to. Can not be null.
- **ActionID:** Identification number of the action that created this belief. Can not be null.
- **ObjectID:** Identification number of the object this belief refers to. Can not be null.
- **Attribute:** Name of the attribute this belief refers to. The attribute does not necessarily have to be listed in any part of the object. For example, it can be a body part. Can not be null.
- **Is:** New / Added value to the attribute this belief refers to. Can not be null.

3.7. Story_Goal

This table holds all the goals happening on a story. They are created through actions in the story. Goals hold the final concept an object wants to reach, its current state and the steps it must take to reach that concept.

3.7.1. Column Descriptions

- **Instance:** Instance number of the goal. Primary Key to the table.
- **ID:** Identification number of the goal. Can not be null.
- **StoryID:** Identification number of the story this goal belongs to. Can not be null.

- **ActionID:** Identification number of the action that modified this goal and, thus, created this instance. In first instances, it refers to the action that created this goal. Can not be null.
- **State:** Current state of the goal. Feasible, unfeasible, completed, etc. Can not be null.
- **Role:** Role of this goal in the story. User primarily for some master plots.
- **Steps:** List of concepts that must be fulfilled to complete the goal. Last step is the goal. Can not be null.
- **Done:** List of steps fulfilled until this instance. Can be null.
- **Identities:** List of identities this goal is known by in the story.
- **ActionPerformed:** Action performed to this goal. Can not be null. In first instances, it will be *Created* or similar.
- AffectedField: Name of the modified field by this instance.

3.8. Story_Object

This table holds all the objects of a story. They are created when they first appear in a story, through an action.

3.8.1. Column Descriptions

- **Instance:** Instance number of the object in this story. Primary Key to the table.
- **ID:** Identification number of the object. Can not be null.
- **StoryID:** Identification number of the story this object belongs to. Can not be null.
- **SequenceID:** Identification number of the story point this object was modified in. Can not be null.
- **ActionID:** Identification number of the action that modified this object and, thus, created this instance. In first instances, it refers to the action in which this object appeared for the first time. Can not be null.
- **StoryRole:** Role of the object in the story. Usually: Protagonist, Antagonist, Secondary. Should not be modified.
- **ThematicRoles:** Thematic roles that this object can perform. Can be null.
- **Goals:** List of identification numbers of the goals this object pursues.

- **At:** Identification number of the location of this object is at in this moment of the story.
- **Age:** Age of the object at this point of the story.
- **Form:** Name or identification number of the object form this object is taking at this point in the story. For example: human.
- **States:** List of states of this object at this point in the story. For example: asleep, dead.
- **Attributes:** List of attributes of this object. For example: pretty, tall.
- **Traits:** List of traits that define this object. For example: good, bad, afraid.
- **Beliefs:** List of the identification numbers of the beliefs this object has.
- **Inventory:** List of the identification numbers of the objects this object has in its inventory.
- **Identities:** List of identities this object is referred as in the story.
- **ActionPerformed:** Action performed to this object. Can not be null. In first instances, it will be *Created* or similar.
- **AffectedField:** Name of the modified field by this instance.

3.9. Story_Relation_Actions

The table containing all action relations. This table represents which actions triggered which, allowing for a deeper analysis of the story.

3.9.1. Column Descriptions

- **ID:** Identification number of the relation. Primary Key to the table.
- **Action1:** Identification number of the first action being part in this relation.
- **Relation:** String defyning the type of the relation of these two actions.
- **Action2:** Identification number of the second action being part in this relation.

3.10. Story_Relation_Beliefs

The table containing all belief relations. This table serves to represent an string of ideas. For example:

The child finds the witch ugly because of her huge claws.

The witch is ugly would be the first belief.

Because would be the relation.

The witch has big claws would be the second belief.

The ID to this relation would be stored in the object *child*.

3.10.1. Column Descriptions

- **ID:** Identification number of the relation. Primary Key to the table.
- **Belief1:** Identification number of the first belief being part in this relation.
- **Relation:** String defyning the type of the relation of these two beliefs.
- **Belief2:** Identification number of the second belief being part in this relation.

3.11. Story_Relation_Objects

The table containing all the object relations. A relation can have a top – lower nature (f.e., father of), in which the first object would be the owner of the relation to the second, or an equal relation (f.e., sibling of), in which both objects share the same status..

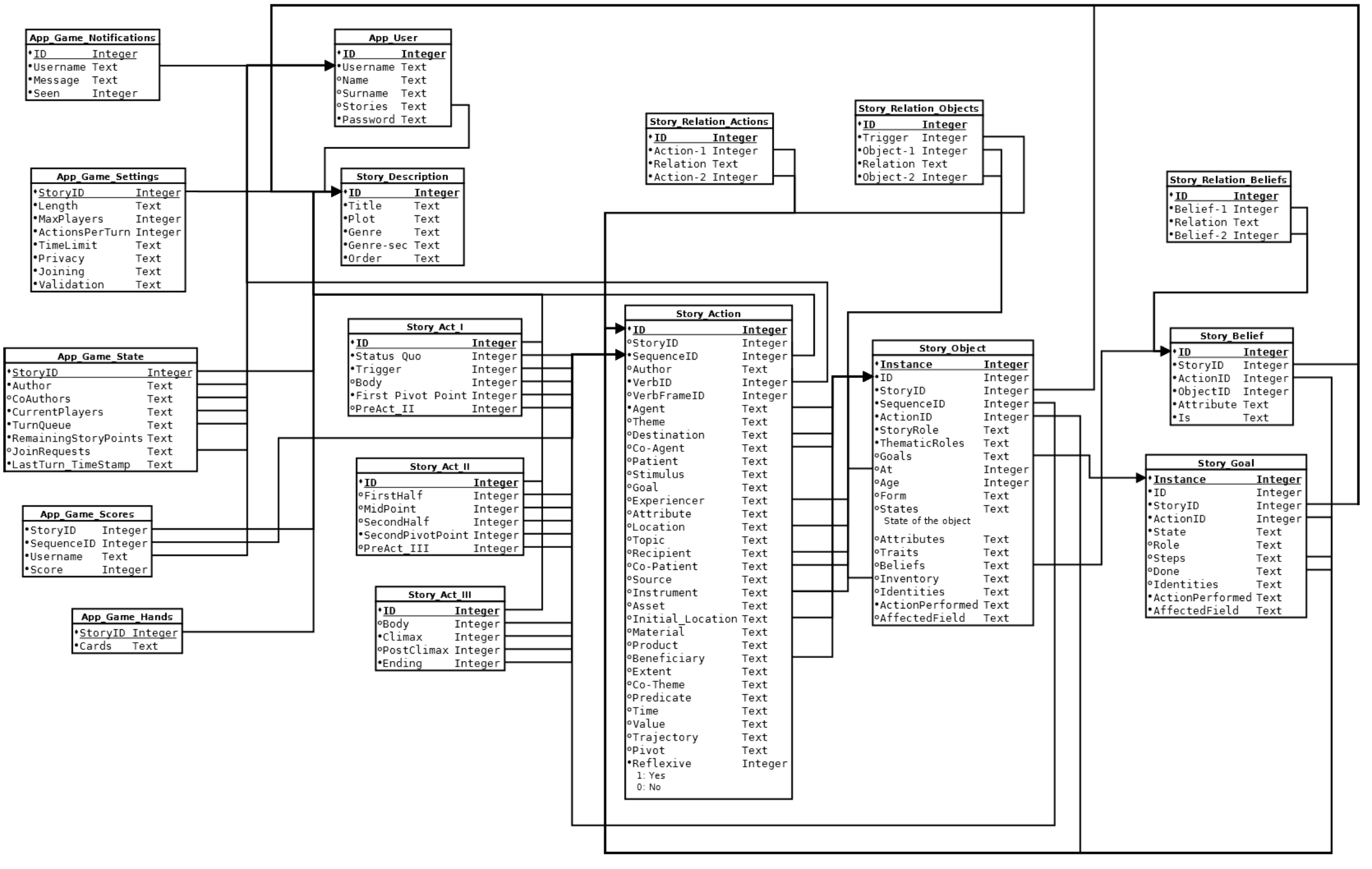
3.11.1. Column Descriptions

- **ID:** Identification number of the relation. Primary Key to the table.
- **Object1:** Identification number of the first object being part in this relation.
- **Relation:** String defyning the type of the relation of these two objects.
- **Object2:** Identification number of the second object being part in this relation.

Anexo F: Esquema final de StoryDB

Este documento fue originalmente exportado a una imagen PNG y ha sido impreso en formato PDF para facilitar la lectura de esta memoria.

La imagen original está disponible para cualquiera que la solicite.


App_Game_Notifications
ID Integer
Username Text
Message Text
Seen Integer
App_User
ID Integer
Username Text
Name Text
Surname Text
Stories Text
Password Text
App_Game_Settings
StoryID Integer
Length Text
MaxPlayers Integer
ActionsPerTurn Integer
TimeLimit Text
Privacy Text
Joining Text
Validation Text
Story_Description
ID Integer
Title Text
Plot Text
Genre Text
Genre-sec Text
Order Text
Story_Relation_Actions
ID Integer
Action-1 Integer
Relation Text
Action-2 Integer
Story_Relation_Objects
ID Integer
Trigger Integer
Object-1 Integer
Relation Text
Object-2 Integer
Story_Relation_Beliefs
ID Integer
Belief-1 Integer
Relation Text
Belief-2 Integer
App_Game_State
StoryID Integer
Author Text
CoAuthors Text
CurrentPlayers Text
TurnQueue Text
RemainingStoryPoints Text
JoinRequests Text
LastTurn_TimeStamp Text
Story_Act_I
ID Integer
Status Quo Integer
Trigger Integer
Body Integer
First Pivot Point Integer
PreAct_II Integer
Story_Act_II
ID Integer
FirstHalf Integer
MidPoint Integer
SecondHalf Integer
SecondPivotPoint Integer
PreAct_III Integer
Story_Act_III
ID Integer
Body Integer
Climax Integer
PostClimax Integer
Ending Integer
Story_Action
ID Integer
StoryID Integer
SequenceID Integer
Author Text
VerbID Integer
VerbFrameID Integer
Agent Text
Theme Text
Destination Text
Co-Agent Text
Patient Text
Stimulus Text
Goal Text
Experiencer Text
Attribute Text
Location Text
Topic Text
Recipient Text
Co-Patient Text
Source Text
Instrument Text
Asset Text
Initial_Location Text
Material Text
Product Text
Beneficiary Text
Extent Text
Co-Theme Text
Predicate Text
Time Text
Value Text
Trajectory Text
Pivot Text
Reflexive Integer
1: Yes
0: No
Story_Object
Instance Integer
ID Integer
StoryID Integer
SequenceID Integer
ActionID Integer
StoryRole Text
ThematicRoles Text
Goals Text
At Integer
Age Integer
Form Text
States Text
State of the object
Attributes Text
Traits Text
Beliefs Text
Inventory Text
Identities Text
ActionPerformed Text
AffectedField Text
Story_Belief
ID Integer
StoryID Integer
ActionID Integer
ObjectID Integer
Attribute Text
Is Text
Story_Goal
Instance Integer
ID Integer
StoryID Integer
ActionID Integer
State Text
Role Text
Steps Text
Done Text
Identities Text
ActionPerformed Text
AffectedField Text
App_Game_Scores
StoryID Integer
SequenceID Integer
Username Text
Score Integer
App_Game_Hands
StoryID Integer
Cards Text

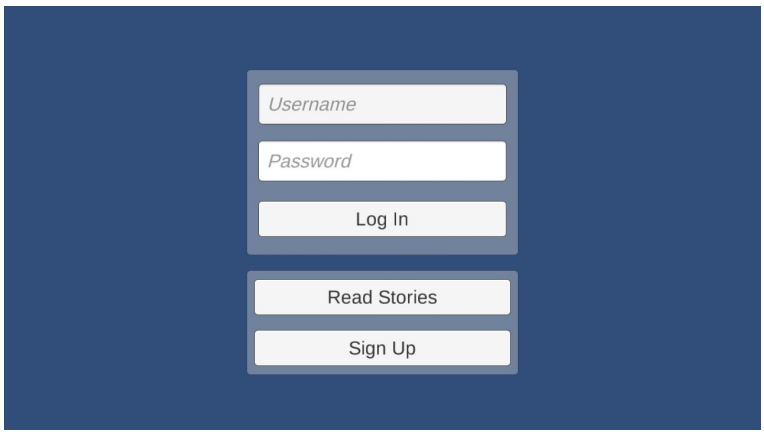
Anexo G: Manual de Usuario de TaleBox

TaleBox: What is it?

TaleBox is a turn-based card game for mobile devices about story creation. The way is simple: the player gets a number of cards at the beggining of her/his turn, with which s/he will have to play the next turn, adding an action to the story.

LOG IN

The first thing shown upon opening the app is this screen:

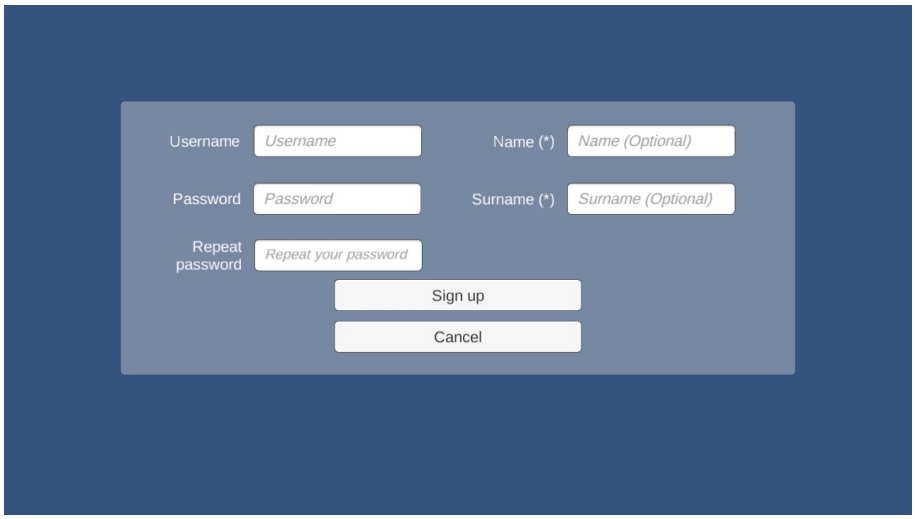


Here, the user can either log in by using her/his account's username and password, or sign up to the app by pressing the *Sign up* button.

The *Read Stories* button should allow the anonymous user to read other players' stories, but that is a feature yet to be implemented.

SIGNING UP

This is the *Sign Up* panel:



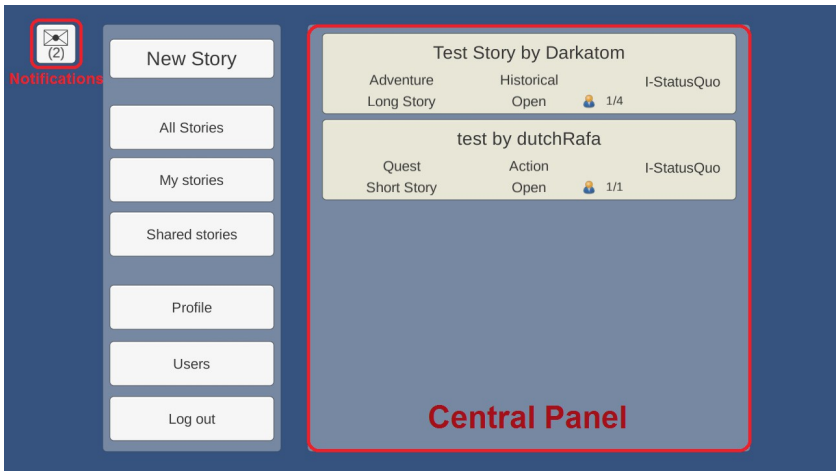
In order to sign up, the user must fill the blanks next to *Username* and *Password*, as well as *Repeat Password* (to ensure the password was correct), which are the only mandatory fields.

Name and *Surname* fields are optional. They will only be displayed on the profile of the user.

Then, press *Sign up*.

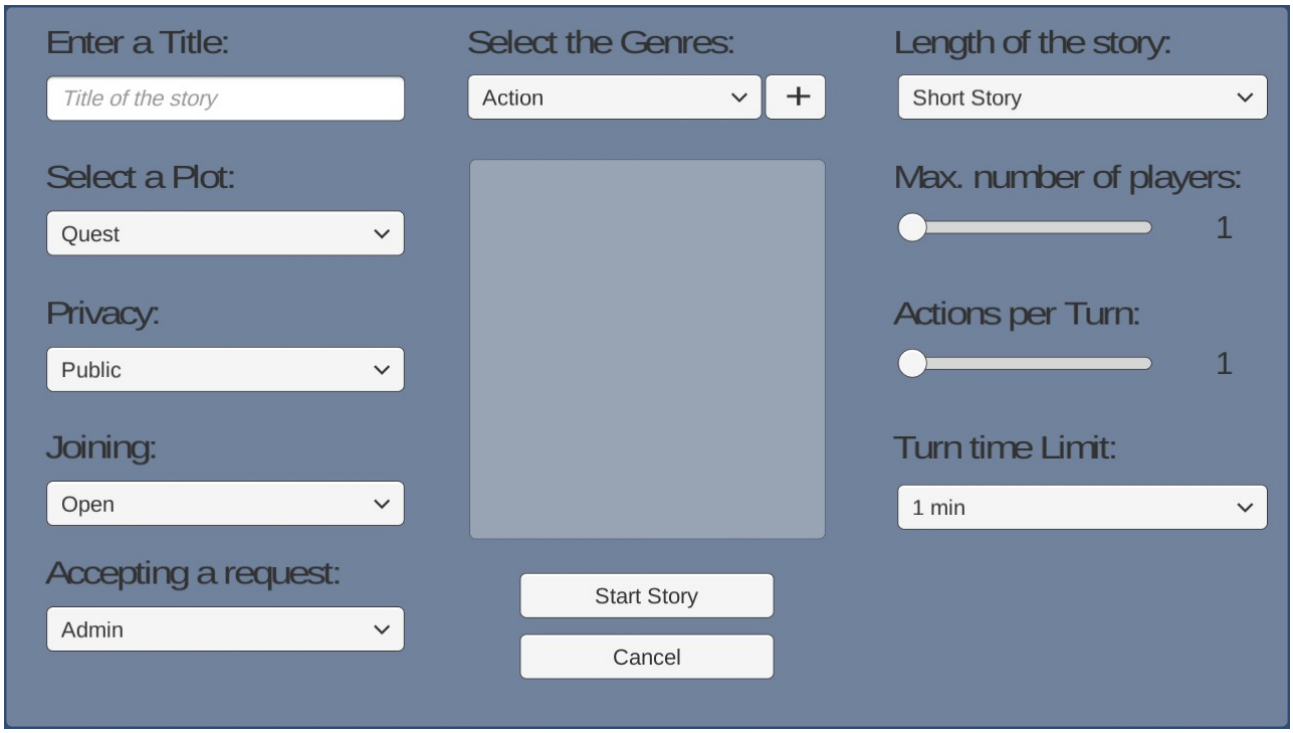
USER PANEL

After successfully logging in the application, the user will be presented with the User Panel. This is the central headquartes of the player. Let's take a look at it:



- ***Notifications:*** Each time a turn is missed or there's a new turn, a message will arrive. To view the pending messages, press this button. A list will appear and a *Back* button. Messages are erased when *Back* is pressed.
- ***New Story:*** Takes the player to the menu that allows to start a new story.
- ***All Stories:*** Shows in the *central panel* all current stories in the system, both finished and on-going. Tapping in one of the stories will take the user to the gameplay screen, where s/he will be able to read the story and play a new turn, if it is available to her/him.
- ***My Stories:*** Shows in the *central panel* all stories that were started by the player, both finished and on-going, even if the player has abandoned the story.Tapping in one of the stories will take the user to the gameplay screen, where s/he will be able to read the story and play a new turn, if it is available to her/him.
- ***Shared stories:*** Shows in the *central panel* all stories the player is currently playing, both started by her/him and others.Tapping in one of the stories will take the user to the gameplay screen, where s/he will be able to read the story and play a new turn, if it is available to her/him.
- ***Profile:*** Shows the profile of the player; name, surname, number of started stories, number of played stories, the average porcentage of missed turns and the number of abandoned stories. These data is shown to help other players decide if they want to invite a player into their stories.
- ***Users:*** Shows the list of users in the system. Tapping in one of their names will show their profiles.
- ***Log out:*** Logs out and shows the log in panel.

NEW STORY



Here, the player can set up a new story.

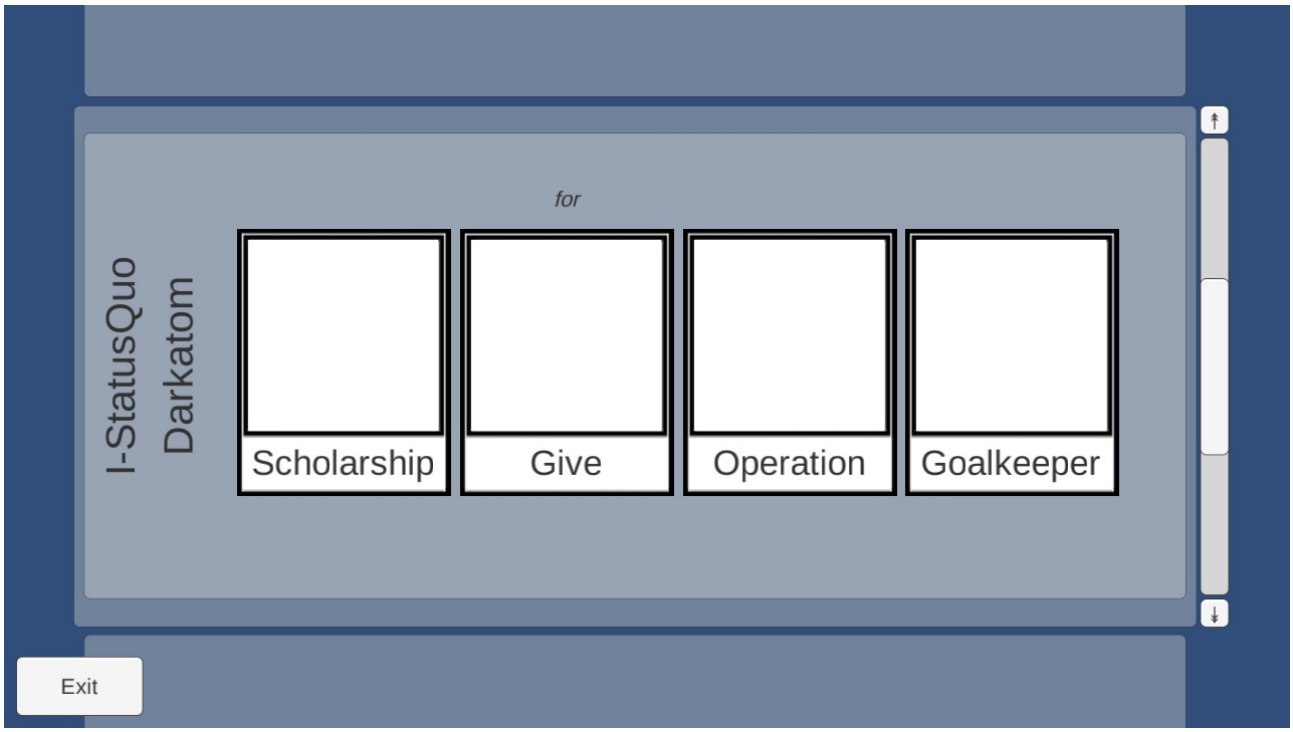
- ***Title***: A name for the story.
- ***Genre***: Choose a genre for the story from the given list. This will impact in the selection of cards dealt by the server to fit in the genre of the story.
- ***Length***: Choose a length for the story from the given list. This will impact in the number of rounds for each story point.
- ***Plot***: Choose a plot for the story from the given list. This will impact in the selection of events given by the server to fit the topic of the story.
- ***Privacy***: Choose a privacy setting for the story. Public stories can be read by anyone.
- ***Joining***: Choose the joining settings for the story.
  - ***Open***: Anyone can ask to join the story.
  - ***Restricted***: Only friends of the current players can request to join the story.
  - ***Closed***: No one can request to join the story. Players can still be invited to join.
- ***Accepting a request:*** Choose how to deal with joining requests.
  - ***Admin***: The administrator of the story (the author by default, the next player in line if the author left, and so on) decides if the request is accepted or declined.
  - ***Poll***: The users get to vote on the request. Majority of positive votes accepts the request.
  - ***Automatic***: All requests are accepted.
- ***Max. number of players***: Choose the maximum number of players that can concurrently play the story.
- ***Actions per turn:*** Choose how many actions can a player submit by turn.
- ***Turn time limit***: Choose the time limit a player has to submit her/his turn from the given list.

Pressing *Start Story* will start the story and take the user to the gameplay screen.
Pressing *Cancel* will take the user back to the User Panel. The story will not be created.

GAMEPLAY SCREEN

In TaleBox, stories are represented as a string of actions (or *moves*) from the beginning to its current state.
The gameplay screen shows both the already played moves and a new turn, should it be available to the user. Moves are represented as panels (see figure below) the player can scroll through to read the story at all times.

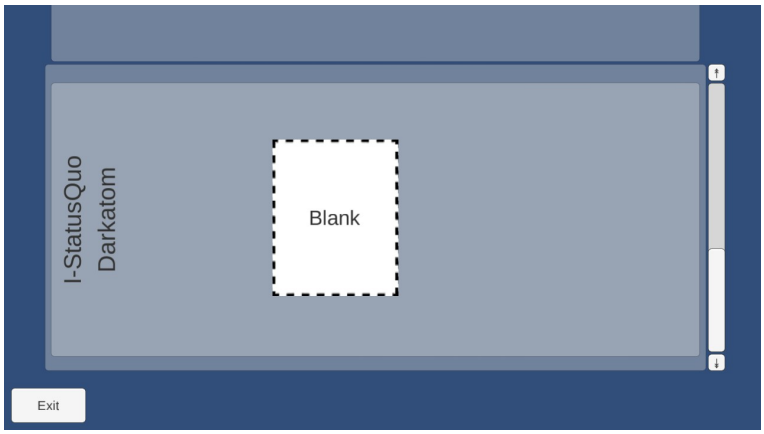
- **Reading the story**



- ***Story Point:*** It is shown on the left side of the panel. The story points are: Status Quo, Trigger, First Pivot Point, Mid Point, Second Pivot Point, Climax and Ending; following the traditional story building structure. The number of rounds to get to the next story point is determined by the length of the story.
- ***Player:*** The author of this move, it is shown right next to the story point.
- ***Cards:*** Each move is shown as a set of cards in line, up to four, forming a sentence.
  - ***Subject*****:** Who does the action. This card can not be left blank.
  - ***Verb*****:** What action takes place. There are all kinds of verbs: emotions, communication, aggression...
  - ***Preposition:*** Some verbs can be used with more than meaning, some of them use prepositions to go with the first and second predicates. For example, one can take something or can take something *from* someone. To facilitate players a better specification of their moves they can use a preposition, given that it's available for the verb chosen. It will be shown over the verb card in italics.
  - ***(First) Predicate:*** First part of the predicate. Here you will find noun, adjective and adverbs cards. Some verbs don't need a predicate (for example: The wizard leaves), so it can be left blank.
  - ***(Second) Predicate:*** Second part of the predicate. Here you will find noun, adjective and adverbs cards. Some verbs don't need a predicate (for example: The wizard leaves), so it can be left blank.

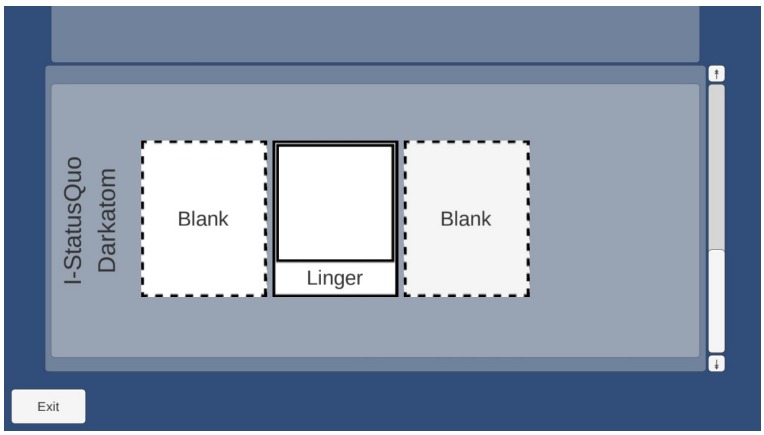
- **Playing a new turn**

Cards are dealt for each turn, which can have several moves. A hand of cards will contain characters and items that have already appeared in the story, as well as new ones and verbs. It must be kept in mind that used cards can not be used again in the same turn, so players should play them carefully.



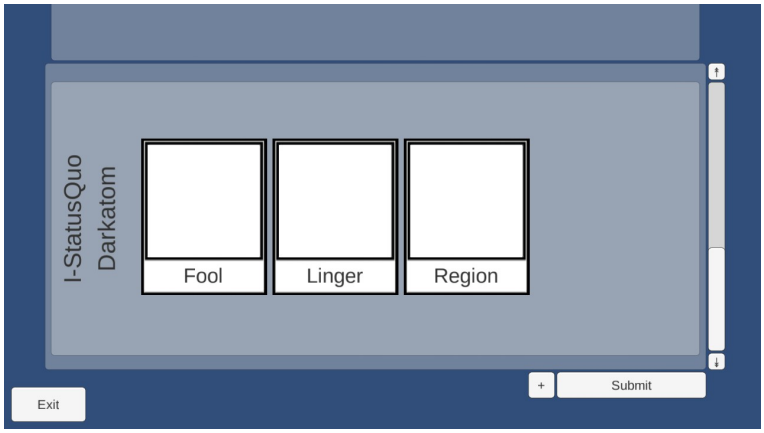
When a new turn is available, a blank move will be shown at the end of the story. The first blank card is for the verb. Tapping on it will show a selection of verbs available (in the hand) for a variety of topics.

The number of verbs given to the player in a hand depends on the number of moves available for the turn, to ensure no player is left without a viable turn.



After choosing a verb, blank spaces for the rest of the sentence will appear next to it. The number of spaces depends on the syntax for the chosen verb and its meaning. For example, *Linger* can have up to two cards, representing the subject and predicate of the sentence.

Prepositions, if available, will be shown in a dropdown list over the verb so the player can adapt the meaning of the sentence to their liking.



Once every blank has been filled with an appropiate card, the *Submit* and *Add a move* buttons will appear.

Add a move only appears if the story allows for more than one move per turn. It will store the latest move and add a new blank one to be filled.

Submit submit all the moves and ends the turn.

Anexo H: Especificaciones del Diseño Gráfico de TaleBox

TALE BOX
GRAPHIC DESIGN

- GUI layout design.

- Card frame.
  - Its colour will depend on the word being represented, so its design must allow changing the colour palette from code.
  - Colours will represent verbs, nouns, adjectives and adverbs.
  - Colour code sugestions for the cards are welcome.

- Card empty placeholder (for when there is space for cards to be placed).

- Icons for cards depending on word classification:
  - **Verb**
    1. Body
    2. Change
    3. Cognition
    4. Communication
    5. Competition
    6. Consumption
    7. Contact
    8. Creation
    9. Emotion
    10. Motion
    11. Perception
    12. Possession
    13. Social
    14. Stative
    15. Weather

  - **Noun**
    1. Abstract
    2. Human
    3. Communication
    4. Machine
    5. Organization
    6. Animal
    7. Location
    8. Body Part
    9. Comestible
    10. Region
    11. Time
    12. Substance
    13. Vehicle
    14. Garment
    15. Sound

- **Adjectives & Adverbs**
  1. Positive
  2. Neutral
  3. Negative

TOTAL:

Card frame + Empty placeholder + 33 icons = 35 coloured drawings.
GUI layout.

Cards should be 640 pixel tall with the standard card proportions.

VERB CATEGORY EXPLANATION

Body	: verbs of grooming, dressing and bodily care
Change	: verbs of size, temperature change, intensifying, etc.
Cognition	: verbs of thinking, judging, analyzing, doubting
Communication	: verbs of telling, asking, ordering, singing
Competition	: verbs of fighting, athletic activities
Consumption	: verbs of eating and drinking
Contact	: verbs of touching, hitting, tying, digging
Creation	: verbs of sewing, baking, painting, performing
Emotion	: verbs of feeling
Motion	: verbs of walking, flying, swimming
Perception	: verbs of seeing, hearing, feeling
Possession	: verbs of buying, selling, owning
Social	: verbs of political and social activities and events
Stative	: verbs of being, having, spatial relations
Weather	: verbs of raining, snowing, thawing, thundering

Anexo J: TaleBox – a mobile game for mixed-initiative story creation

TaleBox – a mobile game for mixed-initiative story creation

Olatz Castaño, Ben Kybartas, Rafael Bidarra
Computer Graphics and Visualization Group
Delft University of Technology
Delft
The Netherlands
olatz.cp@gmail.com, bkybartas@gmail.com, r.bidarra@tudelft.nl

Creating stories appeals to a very wide public, but its computational automation is a very challenging endeavor (Kybartas and Bidarra 2016). We present TaleBox, a mobile game that enables a group of players to collaboratively create their own stories. TaleBox is a mobile card game in which players (authors) take turns to append their contribution to the story, properly combining in a creative way, within the chosen time constraints, the cards that they have been dealt. Cards make for an accessible and dynamic authoring tool.

The game uses a client-server architecture, in which a key role is played at the server by the card *hand generation* system, which tracks the progress of the story and selects for each player a variety of cards that promote both the story coherence and the authors' creativity. This approach strongly benefits from the use of GluNet, a generic, open-source knowledge-base that seamlessly integrates a variety of lexical databases and facilitates commonsense reasoning (Kybartas and Bidarra 2015). By combining, among others, the VerbNet and FrameNet databases, GluNet provides a very sound semantic basis to derive essential semantic relations, that are needed to generate coherent story content. GluNet also provides verbs and generic/abstract words to widen the story space, defined as the group of elements (characters, items, locations, etc.) taking a part in the story. This allows for the generation of a large variety of potential events that are meaningful at a given moment in the story.

After players have chosen a master plot, following the scheme of Tobias (1993), the system is able to determine what kind of major events should happen at each Story Point. These are based on the classical story building structure: Status Quo, Trigger, First Pivot Point, Midpoint, Second Pivot Point, Climax and Ending. In order to ensure coherence, the card hand generation system keeps track of each element's state after an action is performed. Following the notion of the Story Intention Graph (David K. Elson, 2012) and the semantics in GluNet, the system is also capable of assigning goals and other properties to the elements by processing the action. As a result, no actions will be permitted that contradict the current state of any story element, e.g. deploying a character that is dead.

We developed a mapping that associates roles and story genres to the words in GluNet. The system uses the genre to determine which vocabulary can go in the cards being dealt, while roles are checked against the word's semantics, to guarantee that at least one action is possible every turn. Once a set of words is selected, the system checks the story progress and its space. If the story is at a Story Point, a set of significant verbs for events

Proceedings of 1st International Joint Conference of DiGRA and FDG

that match the plot's requirements for that Story Point will be selected, using the verb context and concepts in GluNet. If the story is between Story Points, many other verbs can be selected. The verb cards to be dealt are then randomly chosen from that selection.

The remaining cards, regarding locations, items and characters, will be chosen out of the story space and genre vocabulary selection. For the sake of coherence, the story space has priority, so only elements enabled by their current state can be selected (e.g. the character is not dead). Next, possible gaps of roles in the selected verbs will be filled from the genre vocabulary, so that at least one semantic frame for each verb is covered, thus ensuring an action per turn. Even if the gap was covered with the story state, extra cards will be offered to widen the available choices. Here we show an example card hand:

generated card hand			**possible playable story actions**
settles	peasant	field	peasant settles *on* kingdom
gives	kingdom	bush	peasant gives brother substance
poisons	brother	vegetation	brother poisons field *with* substance
grows	substance	illness	neighbourhood grows field *into* bush
appoints	neighbourhood		kingdom appoints vegetation *to be* illness

Card hands are dealt without distinguishing players, and all cards left after a turn are discarded, to ensure that only usable cards are kept in each player's hand. Players have to come up with the next move using the cards they were handed, so even if they had some plan for certain characters, they still have to creatively find their way using the cards they were given, which can be very enticing and fun.

To make TaleBox even more fun, a score system is kept for each story, rewarding the players who most contribute to it. Additionally, when a Story Point is reached, the player with the highest score is rewarded with choosing the major event in their turn. Major events are very valuable rewards, as they can drastically change the course of a story.

Besides being a game, TaleBox is, in fact, also a powerful mixed-initiative content generation system. Its output – structured and annotated stories, created through a combination of leading human creative work and computer smart authoring assistance – is recorded and stored in an open format. The story database is made available online to the whole community, scientific or otherwise, and can be very useful for a wide variety of applications needing annotated stories, searchable and classified by genre, vocabulary, length, etc…

BIBLIOGRAPHY

Elson, D. K. 2012. "Detecting Story Analogies from Annotations of Time, Action and Agency". In *Proceedings of the LREC 2012 Workshop on Computational Models of Narrative*, Istanbul, Turkey.

Kybartas, B. and Bidarra, R., "A survey on story generation techniques for authoring computational narratives", in IEEE Transactions on Computational Intelligence and AI in Games (2016).

Kybartas, B. and Bidarra, R., "A semantic foundation for mixed-initiative computational storytelling", in Interactive Storytelling, (2015) Springer LNCS 9445, page 162—169.

Tobias, R., 20 Master Plots, Writer's Digest Books; 3rd edition (January 12, 2012). ISBN-13: 003-5313654756.