Automatic generation of Stories with Dialogues

A Final Thesis Document
Presented to
the Faculty of the College of Computer Studies
De La Salle University Manila

In Partial Fulfillment
of the Requirements for the Degree of
Master of Science in Computer Science

by

Alcabasa, Lean Alemania

Ethel Ong
Adviser

August 15, 2011

Abstract

Creative Natural Language Processing (NLP) aims to encompass and model the various aspects of understanding and human creativity in order to design computer systems that can mimic human creativity and provide an engaging interaction with humans. One topic under Creative NLP is Story Generation. As increasing ways of presenting stories are being realized, automatic generation of stories can then find applications in the field of Education and Entertainment. Several existing story generators have already been developed since the 1970s. The existing story generators and dialogue generators focus separately on experimenting with different approaches to generation of narratives, and generation of dialogues between characters. Existing story generators and dialogue generators lack algorithms in creating stories that have both narratives and dialogues. Therefore, the goal of this research is to design and implement an automated story generator that creates stories with narratives and dialogues.

The research produced 9 stories which were presented to 3 evaluators composed of linguists and story writers. The evaluators evaluated the stories based on 3 criterias: Overall Story, Dialogue, and Grammar. Overall story is concerned with the overall quality of the story, Dialogue is more concerned with the quality of the dialogues, and Grammar is for the linguistic / syntactic aspect of the sentences in the story. The Overall Story received a rating of 2.72 out of 4, the Dialogue received a rating of 2.80 out of 4, and the Grammar received a score of 2.52 out of 4. Based on the results of the Overall Story evaluation, the story generator was able to produce acceptable stories with dialogues with a combined overall rating of 2.68 over 4. Based on the results, the approach of mapping the dialogues to the Actions as well as using Utterances to form the Dialogues was able to produce acceptable stories with dialogues. However, there are still improvements that can be made to the research such as improvement to Story Planning, improvement to Overall Story Coherency, improvement to Narrative to Dialogue transition, improve dialogue participant selection, add themes to the stories, researching the use of Personalities to affect Dialogues, inclusion of more types of dialogues, inclusion of more utterances, use of a dialogue history, use of more types of Character Problems, research as an educational tool, and improving Grammar in the stories.

Keywords: natural language processing, story generation, dialogue generation

Table of Contents

List of Figures

List of Tables

1 Research Description

2 Review of Related Literature

3 Theoretical Framework

4 Architecture of the system

In order for the automated story generator to properly generate stories with dialogues, two important elements are needed: the (1) Knowledge Representation and the (2) Story Planner algorithm.

4.1 Knowledge Representation

In order for automated story generation systems to properly create stories, they must have a way of representing the knowledge they need where they get the concepts needed to create the stories. This idea is called the Knowledge Representation or the Knowledge Base.

The knowledge base of the system adapted the concept of **ConceptNet** (Liu & Singh, 2004). ConceptNet is a semantic network made up of concepts / words connected to each other by binary relations such as (dog, isA, animal). ConceptNet was selected because the idea of ConceptNet is easy to adapt and easy to use. However, not all knowledge were presented as binary relations in the implementation of the system. This is because some knowledges need to be presented in a different way in order to serve their purpose.

There are a total of 4 important Knowledge Representations: **Location representation**, **Character representation**, **Object representation**, and **Action / Event representation**. Location representation is concerned with the Backgrounds / Setting where the story will take place and the locations of characters and objects in the story. Character representation is concerned with the Characters in the story. Object representation is concerned with the objects / concepts found in the story. Finally, Action / Event representation is concerned with the actions / events that form the story.

Figure 4.1 shows an overview of how each Knowledge representation interacts with each other. First, *Characters*, *Objects*, and *Actions* all have a relation to the *Location representation* in the way that each are either located in a Secondary Background location (Characters and Objects) or can be executed in a Secondary Background location (Actions). Second, it can be seen that *Characters* are the ones that performs in *Actions*. By having the character perform the Actions, they are turned to events which form the stories. Lastly, when *Characters* perform *Actions*, the *Action representation* does not provide all the information needed by the Story Planner in order to perform the Action. Therefore, the *Action representation* also uses or gets additional knowledge from the *Object representation*.

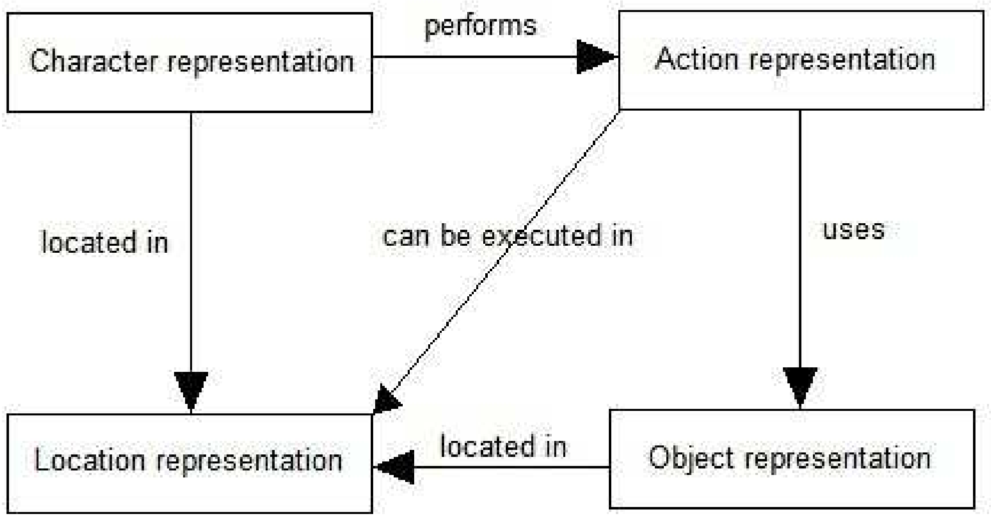

Figure 4.1: Knowledge Representation Interaction

The *Object representation* is able to provide supporting knowledge information to the *Actions* in order for it to be completed.

4.1.1 Location representation

First, the Knowledge Base will contain information about the possible locations a story can take place and the actions, objects located in it. **Table 4.1** show the related relations that are concerned with Location representation:

The relation **hasSecondaryBackground(<PrimaryBackground>, <Secondary Background>)** is used in order to define the parent background of a secondary background. There are 2 types of representation of backgrounds: *Primary backgrounds*, and *SecondaryBackground*. SecondaryBackgrounds are those that are located in a PrimaryBackground and where action / events actually take place. Appendix A shows the available backgrounds in the story.

Next, the relation **locationOf(<Action / Concept>, <SecondaryBackground>)** is used in order to determine where an action takes place, or what objects can be found in a certain location. This relation is used in order to help the Story Planner make the story more realistic by limiting events and objects included in the story based on the current location of the main character using the relation **currentlyInLocation(<Character>,**

Table 4.1: Location representation

Relation name	Description	Format / Example
hasSecondary-Background	Defines a secondary background located in a Primary background	hasSecondaryBackground(<PrimaryBackground> , <SecondaryBackground>) Ex. hasSecondaryBackground(School, Classroom)

locationOf	Defines the secondary background an event / concept can be located in	locationOf(<Action>, <SecondaryBackground>)
		locationOf(<Concept>, <SecondaryBackground>)
		
		Ex. locationOf(eat, cafeteria)
		locationOf(pencil, classroom)

currentlyIn-Location	Defines the current background location of a character in the story	currentlyInLocation(<Character> , <SecondaryBackground>) Ex. currentlyInLocation(Paul, cafeteria)

<**SecondaryBackground**>).

4.1.2 Character Representation

Another knowledge resource in the Knowledge Base is information about the characters. There are 2 types of characters: **Dynamic** and **Static characters**. Dynamic characters are characters that can have many roles in the story and can appear at any time in the story when needed. Dynamic characters consists of *Child characters* and their *Parents.* Additionally, there is a special type of Dynamic Character, the *Main Character* which is the main character in the story. Only child characters can be main characters. Static characters are characters that are found only in specific locations and used to perform specific functions. **Table 4.2**, Table **4.3** and **Table 4.4** shows the relations used to represent the Dynamic characters and Static characters in the story:

The first to be discussed will be the relations used for the *Dynamic characters.* First, the relation **hasMother(<ChildCharacter>, <ParentCharacter>)** and **hasFather(<ChildCharacter>, <ParentCharacter>)** is useful for separating the Child characters from the Parent characters and at the same time, defining both. Next, the relation **hasCharacterProblem(<MainCharacter>, <CharacterProblem>)** is a relation specially used only for the main character. This relation helps define the CharacterProblem that the main character has,

Table 4.2: Dynamic Character Representation

Relation name	Description	Format / Example
hasMother	Defines the mother of a child character	hasMother(<ChildCharacter> , <ParentCharacter>) Ex. hasMother(Paul, Mommy Paula)
hasFather	Defines the father of a child character	hasFather(<ChildCharacter> , <ParentCharacter>) Ex. hasFather(Paul, Daddy Carlo)
hasCharacterProblem	Defines the character problem of the main character	hasCharacterProblem(<MainCharacter> , <CharacterProblem>) Ex. hasCharacterProblem(Paul, Hunger)

Table 4.3: Static Character Representation

Relation name	Description	Format / Example
hasJob	Defines the job of a static character	hasJob(<StaticCharacter>, <Job>) Ex. hasJob(Lady, Teacher)
locationOf	Defines the station / location of the static character	locationOf(<StaticCharacter> , <SecondaryBackground>) Ex. locationOf(Lady, Classroom)

Table 4.4: General Character Representation

Relation name	Description	Format / Example
currentlyInLocation	Defines the current location of a static character	currentlyInLocation(<Character> , <SecondaryBackground>) Ex. currentlyInLocation(Lady, Zoo)
gender	Defines the gender of the character	gender(<Character> , <Gender>) Ex. gender(Lady, Female)
<Character>	Defines the type of character	<MainCharacter> , <ChildCharacter> , <ParentCharacter> , <StaticCharacter>

Table 4.5: Character Problems

<CharacterProblem>	Description
Hunger	The main character is hungry and needs food
Knowledge	The main character needs / wants to know something
Feel(Emotional, Physical)	The character wants to feel something (ex. happiness)

which is defined by the user input to be discussed later. The idea for these character problems came from reviewing the work of Meehan (1977). Only one character problem was retained (Hunger) from those specified of (Meehan, 1977). The idea of character problems is to use the character problem of the main character and how he/she solves it in order to create a story.

Each character can only have one problem throughout the story. There are a total of 3 character problems currently available in the story: Hunger, Knowledge, and Feel. **Table 4.5** shows the problems with descriptions.

The Knowledge problem specifies that the main characters wants to improve his/her knowledge by learning more about different things. The Hunger problem specifies that the character is hungry and wants to eat food. Finally, the Feel problem specifies that the character suffer from a negative Emotional or Physical feeling and wants to solve this problem. The character problems are the main driving force for the story generator to create a story; The idea of character problems is to use the character problem of the main character and how he/she solves it in order to create a story.

The next to be discussed is the *Static character representation.* The relation **hasJob(<StaticCharacter>, <Job>)** is used in order to help define the purpose of the Static character in the story. The **locationOf(<StaticCharacter>, <SecondaryBackground>)** relation is used in order to determine the initial Secondary Background location of a Static Character in the story. This relation is used in order for specific actions / events in the story that needs to be performed by a Static Character in the location.

Finally, the last representation to be discussed is the *General Character representation* which contains representations used in order to represent both the Dynamic Characters and Static Characters. The first relation is the **currentlyInLocation(<Character>,<SecondaryBackground>)**. This relation is used for the relation **requiresMinimumCharacters** to check if it is satisfied, as well as to find a **DialogueSpeaker / DialogueHearer** in the location. The next relation is the **gender(<Character>, <Gender>)** relation which is used by the surface realizer to provide the proper pronoun representations of a character in the story.

4.1.3 Object representation

This section discusses about how Objects and other Concepts are represented in the Knowledge Base. **Table 4.6** shows the relevant relations.

In order to make the selection of properties of concepts more dynamic, the **PropertyOf(<Concept>,<PropertyType>** relation was designed differently compared to the binary relations. Instead of the propertyOf binary relationship, concepts in the knowledge base all have data corresponding to the <PropertyType> they can have such as *isAppearanceLiving, isAppearanceObject, isSize, isAdverb, isQualityLiving, isQualityObject, and isQualityFood. isAppearanceLiving* is associated with modifiers that define appearance of living things such as 'beautiful' or 'handsome'. *isApperanceObject* is used to define appearance of non-living things such as 'shiny'. *isSize* is associated with modifiers that define size such as 'big' or 'tall'. *isAdverb* is associated with adjectives that are applicable to verbs (ex. quick, slow). *isQualityLiving* is associated with adjectives defining a quality of a living thing (ex. kind, honest). *isQualityObject* is associated with the adjectives that define the quality of a non-living thing (ex. hard, smooth). Finally, isQualityFood is used to define the qualities that a food can have. Finally, the **Modifier(<ModifierName>,<PropertyType>** relation is used in conjunction with the **propertyOf** relation. The **Modifier** relation contains the adjectives that can be applied to the PropertyTypes.

Table 4.6: Object representation

Relation name	Description	Format / Example
isA	A relation used to define Concept A as a sub-class of Concept B	isA(Police Officer,Job)
PartOf	A relation used to define Concept A as part of Concept B	partOf(Wheels,Car)
MadeOf	A relation used to define Concept A as being made up of Concept B	madeOf(bottle,plastic)
CapableOf	A relations used to define the actions that a character can apply to the Concept A	capableOf(food,eat)
Conceptually-RelatedTo	A relation used to define the concepts related to a certain Concept A	conceptuallyRelatedTo(wedding , bride)
canDo	A relation used to define the actions Concept A can do	canDo(lion,roar)
howTo	A relations used to define the concepts that are related on how to do a certain Concept A	howTo(chop,peel)
performedWith	A relation used to define the concepts used in performing a certain Action A	performedWith(basketball,ball)

Finally, the **ColorProperty(ball,red)** is another property but stored in a separate table with the same design of the binarily connected relations. The reason for this design of the color properties is because concepts may sometimes have only a specific color. For example, "vegetables" can only be the color green. However, the toy "ball" can have many color. This is so that the knowledge base will be populated with the right color of the concepts.

The **locationOf(tv,living room)** relation is used in order to specify the possible locations of a given concept in the story backgrounds. It is generally used by the story generator to limit the objects that can be found on the backgrounds and try to make the story more believable.

Finally, the relation **isGeneralNoun(Job,isA)** are used to provide additional information that will be used by the story planner in Inquiry Dialogues. This is to give the story planner an idea of what the Inquiry dialogue will be discussing about the chosen topic.

Table 4.7: Object representation

Relation name	Description	Format / Example
PropertyOf	Contains the possible properties that a certain Concept can have.	PropertyOf(<Concept> , <PropertyType> Ex. propertyOf(ball,size)
Modifier	Contains the concepts used to define other concepts	Modifier(<ModifierName> , <PropertyType> Ex. Modifier(big,size)
ColorProperty	Contains a binary relation of Color Properties that a concept can have	ColorProperty(ball,red)
locationOf	Contains the location of the concept / object	locationOf(tv,living room)
isGeneralNoun	Contains additional information of the object / concept that will be used by the story planner in Inquiry Dialogues	isGeneralNoun(Job , <GeneralNounType>)
<PropertyType>	Contains the property types a concept can have	isAppearanceLiving , isAppearanceObject , isSize , isAdverb , isQualityLiving , isQualityObject , isQualityFood
<General-NounType>	Contains the property types a concept can have	isA , partOf , madeOf

Table 4.8: Character Actions

Action	Description	Examples
General Action	Any other action not falling to the other 3 categories	Play Watch tv Clean
PTrans	Transfer of Physical location of character X to location Y	Run, Walk
MTrans	Transfer of information between characters, an object to a character or within a character	Read (object to character) Tell (character to character)
ATrans	Transfer of possession of an object Y from person X to person Z	Give Take Buy

4.1.4 Action / Event representations

Actions are the actions that can be done by each character. The actions that a character can do was helped formed by reviewing the work of Schank (1975). Character actions are also primarily used to create the story events. In this case, Events mean that the main character performs the action specified. There are a total of 4 classifications of actions that characters can do. Each action then has its own relations that it uses in order to represent it. **Table 4.8** shows the actions that a character can do.

The following sub-sections will discuss more about how each type of actions is represented by the system.

General Action representation

The first to be discussed will be the **General Actions** that a character can do. **Table 4.11** shows the relevant relations used to represent the **General Actions**.

The **General Action** defines the general actions that the main character can do in the story. *Events* in this case are defined as the main character performing the "Action" specified along with its various columns. The **locationOf(<Action>, <SecondaryBackground>)** is used in order to limit the possible events that the Story Planner will select based on the background input by the user. The next relation is the **triggersDialogue(<Action>, <DialogueType>)** relation which defines if the action triggers a dialogue between characters. This column can contain four (4) major types of dialogues

Table 4.9: General Action Representation

Relation name	Description	Format / Example
locationOf	Defines the location the action / event takes place	locationOf(<Action>, <SecondaryBackground>) Ex. locationOf(have class, Classroom)
triggersDialogue	Defines the dialogue triggered by the action	triggersDialogue(<Action>, <DialogueType>) Ex. triggersDialogue(discuss, Inquiry)
createCharProblem	Defines the character problem created by the action / event	createCharProblem(<Action>, <CharacterProblem>) Ex. createCharProblem(play, Hunger)
solveCharProblem	Defines the character problem solved by the action / event	solveCharProblem(<Action>, <CharacterProblem>) Ex. solveCharProblem(eat, Hunger)
requires-MinimumCharacters	Defines the minimum characters needed by the action / event	requiresMinimumCharacters(<Action>, <MinimumCharacter>) Ex. requiresMinimumCharacters(play, 2)
dialogueSpeaker	Defines the dialogue speaker in the action / event	dialogueSpeaker(<Action>, <Character>) Ex. dialogueSpeaker(play, DynamicCharacter)
dialogueHearer	Defines the dialogue hearer in the action / event	dialogueHearer(<Action>, <Character>) Ex. dialogueHearer(play, StaticCharacter)
isNegotiationEvent	Defines if the action / event can be used for negotiation	isNegotiationEvent(<Action>, <Character>) Ex. isNegotiationEvent(play, Child)
hasNextEvent	Defines the next event of an action / event	hasNextEvent(<Action>, <Action>) Ex. hasNextEvent(get hurt, cry)

hasRequirement	Defines the requirement needed by the action / event	hasRequirement(<Action>, <Requirement>)
		
		Ex. hasRequirement(play, askDo)
		
		hasRequirement(eat, NeedObject(<object>)

hasPerforms	Defines if the action / event performs a special event	hasPerforms(<Action>, <Perform>)
		
		Ex. hasPerforms(go shopping, Prox1(Mall))
		
		hasPerforms(cry, Incapacitate)

hasPatiens	Defines the patiens the action / event acts upon	hasPatiens(<Action>, <Patiens>) Ex. hasPatiens(watch, tv)
hasFirstSubEvent	Defines an action / event that precedes the action / event	hasFirstSubEvent(<Action>, <Action>) Ex. hasFirstSubEvent(act in play, learn script)
isTopicGetter	Defines if an action / event that precedes the action / event is a topic getter	isTopicGetter(<Action>) Ex. isTopicGetter(learn)

Table 4.10: General Action Representation

Relation name	Description	Format / Example
isLocationSpecific	Defines if a Topic Getter action is location specific or not	isLocationSpecific(<isTopicGetter(<Action>)>, <boolean>) ex. isLocationSpecific(learn, false)
isDiscussion	Defines if a Topic Getter action is a discussion or not	isDiscussion(<isTopicGetter(<Action>)>, <boolean>) ex. isDiscussion(learn, true)

Table 4.11: General Action Representation

Relation name	Description	Value
<Action>	Defines the name of the action	string ex. discuss
<DialogueType>	Defines the type of dialogue	Information-Seeking , Inquiry , Deliberation , Persuasion
<Minimum-Character>	Defines the minimum number of characters required	integer
<Requirement>	Defines the requirement needed	askDo , NeedObject(<Concept>) , Problem(<CharacterProblem>)
<Perform>	Defines the special perform of an action	Incapacitate , Prox1(<PrimaryBackground> , <SecondaryBackground>) , Prox2(<PrimaryBackground> , <SecondaryBackground>)
<Patiens>	Defines the patiens of an action	string

namely the Information-Seeking dialogue, the Inquiry dialogue, the Deliberation dialogue, and the Persuasion dialogue, which will be discussed in a later section. The next relation **createCharProblem(<Action>, <CharacterProblem>)** is used in order to find events that will be used to create the CharacterProblem. The **isNegotiationEvent(<Action>, <Character>)** specifies if the action can be used as a negotiation in a negotiation dialogue. The <Character> element defines which character type can use that specific action / event for negotiation; it can either be a Parent negotiatable or a Child negotiatable. In real life, parents and children have different natures, for example, parents wanting the child character first to do his/her chores while a child character would want the main character first to play with him/her.

The **hasRequirement(<Action>, <Requirement>)** column indicates requirements needed to perform an event. The possible requirements are *NeedObject(<object>)*, *askDo*, and *Problem(<CharacterProblem>)*. The NeedObject requirement specifies that the main character needs to have the specified object before being able to execute the event. The askDo requirement specifies that the main character should ask the hearer what they are going to do (ex. "What are we going to do?" "We are going to learn"). Finally, the Problem requirement specifies that the event requires that a Character Problem is existing before executing the action.

The **hasPerforms(<Action>, <Perform>)** specifies if the action performs a special function. The possible special functions are *"Incapacitate", "Prox1(<PrimaryBackground> , <SecondaryBackground>)"*, and "Prox2(<PrimaryBackground> , <SecondaryBackground>)". *Incapacitate* means that the main character will be unable to move / act and will need to be approached by another character in order to help him/her. *Prox1 and Prox2* specifies if the action moves characters to another location after the event. Prox1 includes the main character as well as other characters in the location while Prox2 onl moves the main character. The location can either be a Primary Background or a Secondary Background.

The **isTopicGetter(<Action>)** relation helps define if an action is a Topic Getter. TopicGetters are actions that are used in conjunction with MTrans actions / Inquiry dialogues. Inquiry dialogues need a topic to talk about and the TopicGetters are actions that are able to determine a topic for the inquiry dialogue. The *isLocationSpecific(<isTopicGetter(<Action>)>, <boolean>)* relation indicates if the topic to get is location specific (choices are constrained to the current location of the main character). *isLocationSpecific* can have a boolean value of *true* or *false*. Finally, the *isDiscussion(<isTopicGetter(<Action>)>, <boolean>)* is used as support in the realization process. The isDiscussion can contain a boolean value of *true* or *false*. The difference between the two is that *true* adds an ”about” preposition to the topic to be discussed. Examples of TopicGetter actions that has discussion type is the ”learn” action, which will produce something like ”We are going to learn *about* food”. An example of a ”specific” discussion type is the action ”see”, which will produce something like ”We are going to see animals”.

ATrans Actions representation

The next action type to be discussed will be the ATrans action. **Table 4.12** shows the relevant relations used to represent ATrans actions:

The ATrans action is used in order for the main character to acquire an object. It is triggered if an action / event has a ”NeedObject” requirement. The **isATransType(<ATransAction>, <ATransType>)** defines the type of ATrans that the action is. There are two (2) types of ATrans: specificInstance and listOfOptions. specificInstance means that the action is used in order to gain a specific instance of an object and listOfOptions means that the action is used to select from a list of possible options given an object. For example, the object needed is ”Food”. specificInstance already finds an instance of ”food”, for example, ”hamburger” and uses it already on the dialogue (”Can I have a hamburger?”), while listOfOptions triggers a dialogue that specifically allows the receiver to choose from a list of instances of the object, such as

```
"Can I have food?"
"I will give you food"
"What do you give?"
"I give spaghetti, hamburger, and water"
```

hasRequestType(<ATransAction>, <RequestType>) specifies the type of request the ATrans is. There are two (2) possible values for this: *request* or *knowledge*. Request means that the receiver is requesting the giver to give him/her the needed object. Knowledge on the other hand means that the receiver is requesting the giver to tell him/her where he/she can find the needed object. Request triggers a ”Can” IS dialogue, while knowledge triggers a ”Where” IS dialogue. The *Location* is the background location in which the ATrans action can be done. It has a similar format to that of the Actions table.

The **canTriggerNegotiation(<ATransAction>, <boolean>)** relation specifies if the ATrans action triggers a *Negotiation dialogue* if the boolean value is *true*. However, if it is *false*, the ATrans action only triggers an *Information-Seeking dialogue* asking for the item.

Table 4.12: ATrans Action representation

Relation name	Description	Format / Example
isATransType	Defines the ATrans type of the ATrans action	isATransType(<ATransAction>, <ATransType>) Ex. isATransType(buy, specificInstance)
hasReceiver	Defines the receiver of the object in the ATrans action	hasReceiver(<ATransAction>, <Character>) Ex. hasReceiver(buy, MainCharacter)
hasGiver	Defines the giver of the object in the ATrans action	hasGiver(<ATransAction>, <Character>) Ex. hasGiver(buy, StaticCharacter)
hasRequestType	Defines the request type of the ATrans action	hasRequestType(<ATransAction>, <RequestType>) Ex. hasRequestType(ask, request)
locationOf	Defines the location the ATrans action is executed in	hasGiver(<ATransAction>, <SecondaryBackground>) Ex. locationOf(buy, Cafeteria)
canTriggerNegotiation	Defines if the ATrans action triggers a negotiation dialogue	canTriggerNegotiation(<ATransAction>, <boolean>) Ex. canTriggerNegotiation(ask, true)
definesGiveAction	Defines the give action used in the ATrans action	definesGiveAction(<ATransAction>, <GiveAction>) Ex. definesGiveAction(buy, sell)
definesReceiveAction	Defines the receive action used in the ATrans action	definesReceiveAction(<ATransAction>, <ReceiveAction>) Ex. definesReceiveAction(buy, give)
<ATransType>	Defines the ATrans type values	listOfOptions , specificInstance
<RequestType>	Defines the Request type values	request , knowledge
<GiveAction>	Defines the Give action value	string
<ReceiveAction >	Defines the Receive action value	string

Table 4.13: MTrans Action representation

Relation name	Description	Format / Example
isMTransAction	Defines if an <Action> is an MTrans Action	isMTransAction(<Action>) Ex. isMTransAction(discuss)
isMTransType	Defines the MTrans type of the MTrans action	isMTransType(isMTransAction(<Action>), <MTransType>) Ex. isMTransType(discuss, CharHear)
<MTransType>	Defines the MTrans type values	CharHear , CharHear2 , CharAction , CharAction2

MTrans Actions representation

The MTrans action is used in order to defines actions that serve the purpose of knowing more about a certain topic. **Table 4.13** gives the relevant relations used to represent MTrans Actions.

The first relation is the **isMTransAction(<Action>)** which defines if a General Action (<Action>) is an MTrans Action. The actions that are usually MTrans Actions are those that triggers an Inquiry dialogue. The **isMTransType(isMTransAction(<Action>), <MTransType>)** relation specifies the type of MTrans that happens in this action. It provides information of how the story planner is going to handle the MTrans action. There are a total of four (4) MTrans types: CharHear, CharHear2, CharAction, and CharAction2. *CharHear and CharHear2* are MTrans types that define that the knowledge transfer is done by characters speaking to each other / discussing with each other. The only difference is CharHear2 is location specific, meaning that the things they discuss about are limited to what can be found in the location. For an example, take the action "discuss1" as a CharHear and "discuss2" as CharHear2 and the topic is "animals". "discuss1" is located in a classroom and "discuss2" is located in the zoo. "discuss1" is not limited to what the discussion revolves around since it is not location specific. However, "discuss2" discussion is only limited to the animals that can be found in the zoo. The purpose of this is to try to make the story more believable with regards to the type of action / event happening by limiting the objects that can be seen / accessed in the location. *CharAction and CharAction2* means that the knowledge transfer is done by the characters performing the action specified (ex. "read", "see"). CharLocation2 is location specific.

PTrans Actions representation

Finally, the last character action is the PTrans action which is used to define actions that define a change of location of a character. Some examples are ”walk” and ”run”. PTrans actions are used to provide variety to the story when characters move to different locations.

4.2 Story Dialogue Generator

Figure 4.2 shows the proposed architecture of the system. The system takes the following steps in generating a story:

1. The user will need to specify several inputs
2. The inputs will be passed on to the story planner
3. The story planner will continuously generate the sequence of events that will be used to create the story with the aid of the Knowledge Base.
4. While doing this, abstract Sentence Specifications of the events will also be generated by the Story Planner and stored into memory
5. Once all the events are selected and all the sentence specifications are done, the algorithm will then pass the abstract sentence specifications to the surface realizer which will then realize the sentences into English

A detailed description of each part will be discussed in the further sections.

4.3 User Inputs

Each story is created with the help of three (3) user inputs: (1) the location of the story, (2) a character problem, and (3) the lesson / topic of vocabulary acquisition. The general purpose of the background is to help provide a setting for the story to be created. However, this does not mean that the story planner is limited only to this background. It only specifies that the story will sometime take place in the specified background and perform some kind of event there. There are a total of 5 backgrounds where the story can occur: School, Park, Home, Mall, and Zoo.

Character problems are the next input to be discussed. Character problems are problems that characters have that they need to solve by the end of the story. The idea is to use the problem of the main character to create the story by having him/her take the necessary steps in order to solve it. There are a total of 3 character problems currently available in the story: Hunger, Knowledge, and Feel. Hunger is the problem where the character is hungry and needs food. Knowledge is the problem that the character needs / wants to know something. Finally, Feel is the problem that the character wants to feel something (ex. happiness).

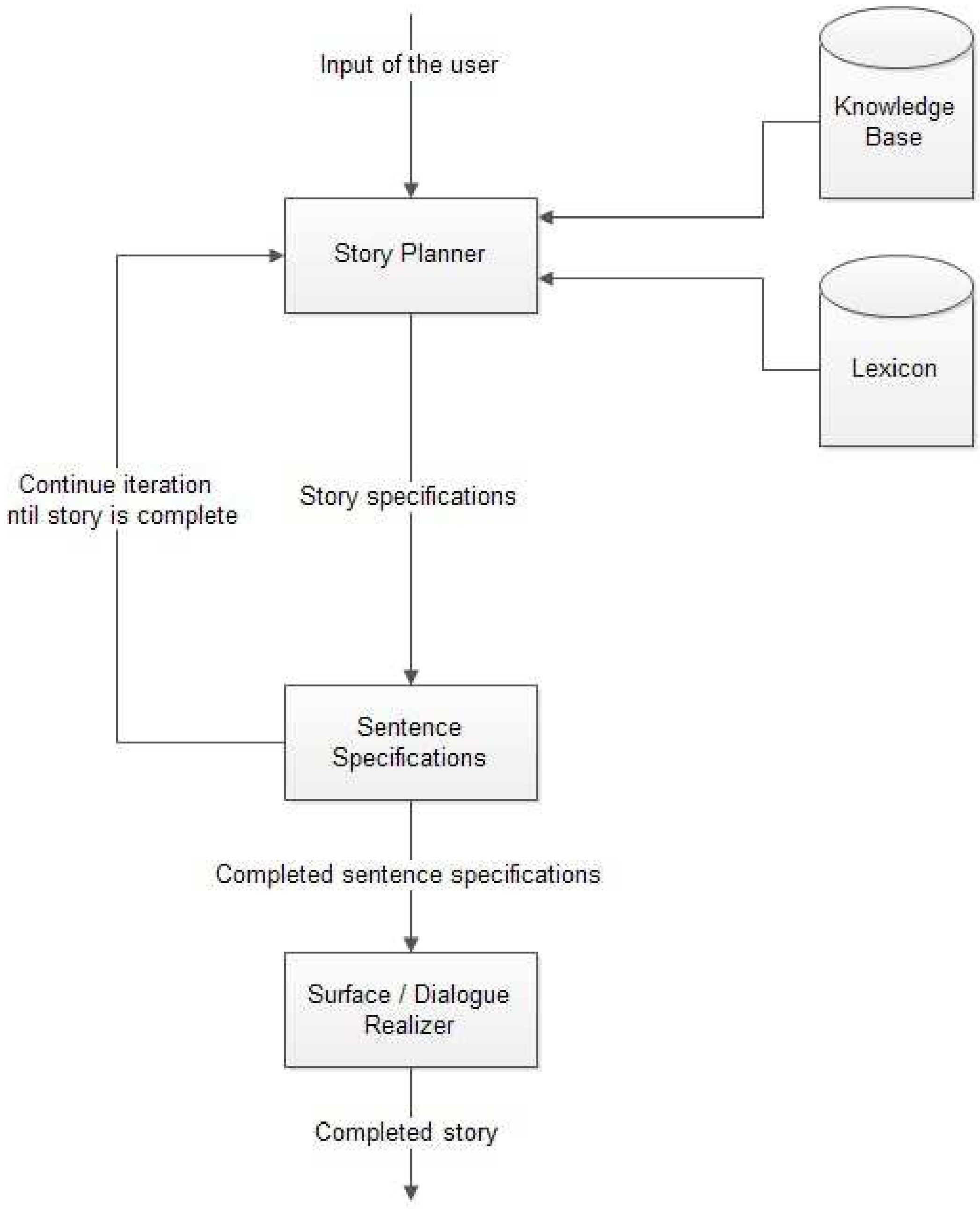

Figure 4.2: Architectural Framework

Table 4.14: Topics

Topic	Sub-topics
Nouns	Types of nouns • People • Things • Animals • Places
Verbs	Tenses of verb • Simple Past Tense • Present Tense • Expression of Future Time
Adjectives	Adjectives Adverbs

Finally, the lesson selected alters the discussion of the characters in the story. The topics were identified and scoped down from the interview given by Correa, R. and Rupinta, J. There are a total of 3 topics: Nouns, Verbs, and Adjectives. Each primary topic then has more specific topics called sub-topics. Sub-topics act as something that is present in the story passively and its main goal is to provide variety to the story. The input the user will enter is the main topic (Noun, Verb, or Adjective). **Table 4.14** give the primary lessons that are to be included in the system. Depending on the lesson selected, they will try to state more related words in their discussion that pertains to the lesson selected by the user.

4.4 Story Planner Module

The inputs will be then passed on to the Story Planner module. The story planner is the one to create the abstract story specifications of how the story will go. The story planner will follow a specific plot structure, specifically, a modified version of the Freytag's Pyramid plot structure when creating stories.

The plot structure is composed of the Exposition, Inciting Incident, Climax, Falling Action, and Conclusion. The notable difference between the original Freytag's pyramid structure is that the Inciting Incident and Rising Action is combined, as well as the Falling Action and the Resolution.

The first plot is the **Exposition**. Exposition is the setting of the story / scene. The next plot is the **Inciting Incident**. The Inciting Incident is an event where something happens to start the action. This event is usually the beginning of the main conflict of the story and is also composed of events that builds up the tension. The actions / events that define the Inciting Incident are those that can create the Character Problems. The middle plot is the **Climax**. The Climax is the moment of greatest tension in the story. There is no direct specification of the Climax in the Knowledge representation used. Instead, what is considered the climax is any requirements / difficulties that the character needs to perform first before being able to reach the Falling Action. The Climax is followed by the **Falling Action**. The Falling Action depicts events that follow as a result of the climax. It also signals that the story is about to end. The Falling action events can be found by finding those that have a specification that can solve the Character Problem. Finally, the final plot is the **Denouenment** or the **Conclusion**. It depicts the final outcome of the story.

The Story Planner module is composed of two main parts: the **Story Planner module itself**, and the **Dialogue Handler**. The Story Planner module itself is more concerned with the plots of the story, as well as the execution of events. The Story Planner module also has a sub-module called the Story Executor which is responsible with executing the actions / events that the Story Planner module has found. The Dialogue Handler is a sub-module called by the Story Planner every time there is a Dialogue triggered by the Story Planner. **Figure 4.3** shows the general framework of the whole Story Planner module.

The sub-sections below will then explain the Story Planning process of the system for each story plot.

4.4.1 Exposition and Inciting Incident

Even though the first plot specified in the plot structure is the **Exposition**, the Story Planner will need to find an event for the **Inciting Incident plot** first. This is to find the very first event that will be formed and then using the specifications of that event, the Story Planner will then form the **Exposition** of the story.

For the selection of potential events for the **Inciting Incident plot**, the Story Planner uses the <CharacterProblem> the user specified and consult the Knowledge Base for events that can create the problem using the **createCharProblem** relation. All actions that fall under this category will be taken into account initially by the Story Planner. However, in order to stay consistent with the background, the Story Planner will filter out the returned Actions that cannot

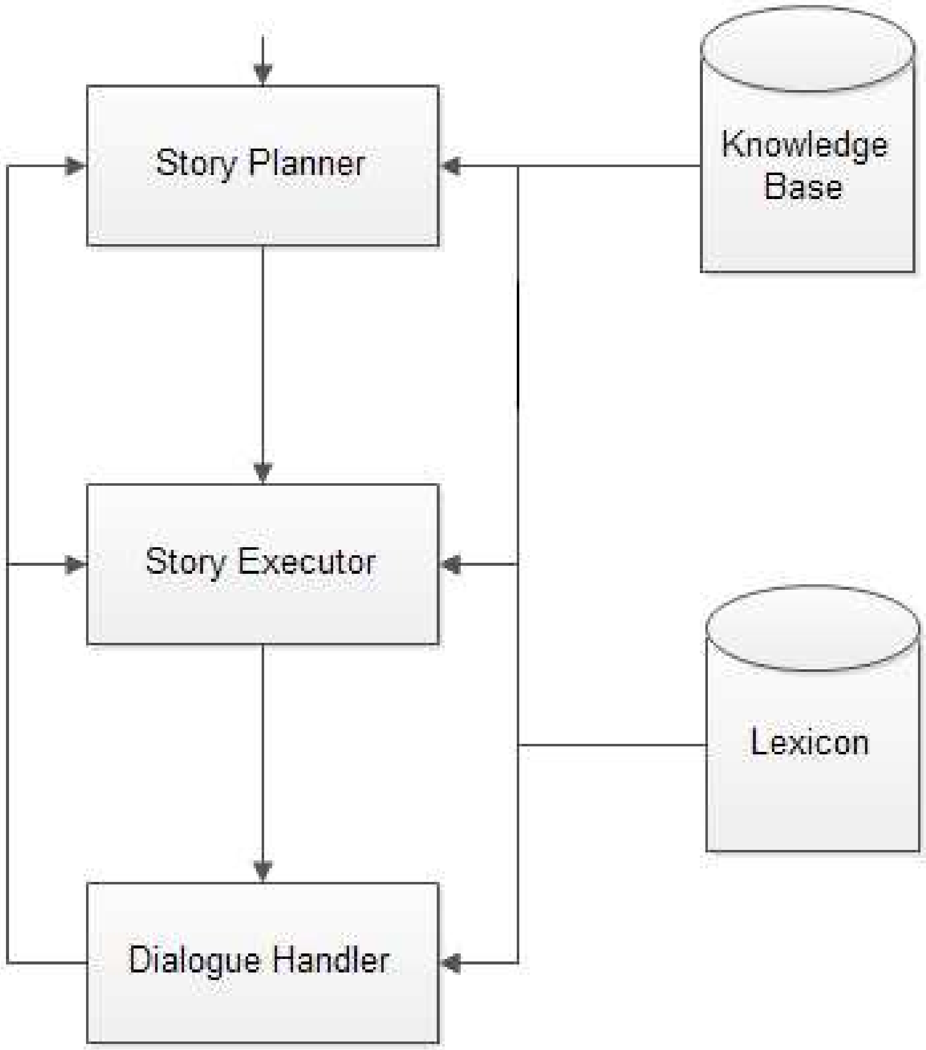

Figure 4.3: Story Planner module

be executed in the <PrimaryBackground> selected by the user using the **locationOf** relation. However, if all the returned actions are filtered out, then the Story Planner determines that the story will need to start in a different setting from the specified Primary Background in order to create the Inciting Incident. The action selected will then be passed onto the Story Executor.

4.4.2 Climax

The **Climax plot** is formed by the Story Planner when it tries to find ways to satisfy the requirements needed by the events of the **Inciting Incident** and the **Falling Action** during the execution of events. The requirements that an event may have is defined by the relations *satisfyMinimumCharacters* and *hasRequirements*.

4.4.3 Falling Action

The **Falling Action plot** is the plot where the Story Planner tries to find the events needed to solve the Character Problem of the main character. It finds a potential Falling Action event using the *solveCharProblem* relation. Similar to the Inciting Incident, it also prioritizes events that can be executed in the current Primary Background. Additionally, however, if the user specified Background is not yet done, then it puts additional priority in selecting events that are located in the user specified Background. If the priorities fail, then it just randomizes from the applicable events.

The selected event will then also be given to the Story Executor to execute.

4.4.4 Conclusion

Finally, the Conclusion sums up the story by having the character go tired and go home. *Sentence Specification Simple Sentences* to represent this are specified by the Story Executor. The main character then recounts the different new things that he/she has learned through the story. In order to keep track of what the main character has learned, a function called *AddSolver* is called every time an Inquiry dialogue, Deliberation dialogue, and Negotiation dialogue occurs and takes notice of what they discussed about. The location of AddSolver in the dialogues was identified through observation because it appears that the three dialogues mentioned provided the best scenario on which the main characters can learn about new things. *Sentence Specification Simple Sentences* to represent this are specified by the Story Executor.

4.5 Story Executor Module

The Story Execution module is sub-module of the Story Planner. It is a function that handles execution of the actions selected by the Story Planner and turns them into events in the story. Below is a pseudocode of the steps the Story Execution module takes in turning an action into an event:

```
storyExecutor(<Action>)
{

     Initialize eventlocation (where the event will take place)

     Execute any FirstEvents

     if <Action> is first event executed
          Call function Exposition

     Add <Action> to executed events

     Move main character to eventlocation

     Satisfy event requirements

     if <Action> triggers any dialogue
          Call function DialogueHandler(dialoguetriggered)
     else
          Call narrative sentence to form event

     if <Action> creates character problem
          Add main character problem
     else if <Action> solves character problem
          Remove main character problem

     Perform any special performs of the <Action>

     if <Action> has a NextEvent
          Call storyExecutor(nextevent)
}
```

The subsequent sub-sections then discuss the pseudocode of the Story Execution Module in more detail.

Event Location determination

The first task of the Story Executor is to determine the <*SecondaryBackground*> of where the action will take place. Since an action can be executed in many places (ex. "see" can be executed in "Animal Room" or "Bookstore"), the Story Executor will first determine the location the action will take place. In finding the location of the story, the Story Executor has some priority algorithm:

```
determineEventLocation(<Action>, locationOf(<Action>),
specific location)
{
If <Action> has specific location specified already
     return specific location
else
    if <PrimaryBackground> has already been went through
        if <Action> can be executed in same location
        as <MainCharacter>
            return location of <MainCharacter>
        else if <Action> can be executed in a <SecondaryBackground>
        that is located in the <PrimaryBackground>
        of the location of the <MainCharacter>
            return <SecondaryBackground>
        else
            return random(locationOf(<Action>))
    else
    {
        find a <SecondaryBackground> located in the
        <PrimaryBackground> that the event can take place

        if <Action> cannot be executed in a
        <SecondaryBackground> located in <PrimaryBackground>
        {
            if <Action> can be executed in same location
            as <MainCharacter>
```

```
                return location of <MainCharacter>
            else if <Action> can be executed in a
            <SecondaryBackground> that is located
            in the <PrimaryBackground>
             of the location of the <MainCharacter>
                return <SecondaryBackground>
            else
                return random(locationOf(<Action>))
        }
    }

}
```

First, it checks if the action to be executed already has an explicit location that it will take place in. This circumstance arises in particular areas such as the Persuasion dialogue.

Next, it checks if the user specified Primary Background has already been went through already. If not, it prioritizes to execute the action in a possible Secondary Background located in the user specified Primary Background. This is to ensure that the story will still happen or go through in the user specified Primary Background.

Next, it checks if the action can be executed on the current location of the main character in order to execute the event there and avoid unnecessary movement of location.

Next, it checks to see if the action can be executed in locations close to the current location of the main character. Closeness means that the location are located under the same Primary Background.

Finally, if all the priorities fail, then the story executor just chooses randomly from the list of available backgrounds that the event can be executed in. If the location chosen by the story executor goes through the user specified Primary Background, then it sets a flag that it has already traversed the user specified Background. This means that all succeeding events can be executed anywhere (other priorities except 3 still apply).

Execution of First Events

The next task of the Story Executor is to execute any first events that the action has. An action may trigger this in many ways.

```
executeFirstEvents(<Action>)
{

     if <Action> has hasFirstSubEvent
          storyExecutor(hasFirstSubEvent(<Action>)

     if <Action> hasRequirement Problem
     {
          find an action that can create the Problem
          storyExecutor(action / event found)
     }

     if <Action> triggers an Inquiry dialogue
     {
          check if Topic for Inquiry dialogue is existing

          if Topic does not exist
               storyExecutor(<Action> that sets a topic)
     }

}
```

It checks if it has a specified first event specified in the Knowledge Base using the relation *hasFirstSubEvent*. If it has a specified first event, then the Story Executor performs that event first by calling itself and providing the necessary inputs of the first event.

Next, if checks if it requires that a Character Problem is already existing using the relation *hasRequirements*. If the Character Problem required is still not existing (using the main character states), then it executes an event first that can create the problem.

Finally, the last possible way to trigger first event is (3) if the action triggers an "Inquiry" dialogue. Since Inquiry dialogues require that a topic be set first, it checks to see if there is already an existing topic that can be talked about. If there is no topic to talk about, it uses the Knowledge Base and finds actions that can create a topic for discussion.

Also, similar to the story executor, it also prioritizes actions that are close to the main character location. Also, in executing the first events, the Story Executor is called to execute the first event specified. In this way, if the first event also has first events, then it will continue to find the very first event that will need to be executed in the chain of events.

Execution of Exposition

The next task that the Story Executor will do is to check if the **Exposition** is already called. The reason that this is where the **Exposition** is located is because the Story Executor determines which is the very first event that occurred and more importantly, where it occured. Using this knowledge, it then properly calls the **Exposition** function to properly set the setting of the story.

```
setExposition(<SecondaryBackground>)
{
      Set weather

      Initialize Characters
      {
            Set <MainCharacter>
            Set <Character> locations
      }
}
```

The first task of the **Exposition** is to set the weather. It does this by consulting the Knowledge Base and finding the concepts *conceptuallyRelatedTo* "weather".

The next task of the Exposition is to initialize the characters in the story. The first step it takes is to get the main character. It does this by getting the Dynamic Characters from the *DynamicCharacters* table in the Knowledge Base. Only child characters can be considered as main characters. The next step would be to initialize the locations of the other characters in the story. This includes the main character, other Child characters, and Parent characters. In setting the location of the characters, the exposition does this by getting the Secondary Backgrounds of the Primary Background that the first event is executed in. For the main character, his/her initial location is already set to the location where the first event occured. For the case of the Dynamic Characters, the exposition randomizes from the Secondary Backgrounds returned and sets the location for each child character as well as parent characters. However, there are special cases here. First, if the Primary Background is "Home", child characters are not initialized in the "Home" of the main character. Rather, they will be randomized in the Secondary Backgrounds of "School" or "Park". The second special case is with regards to Parent Characters. If the Primary Background of the first event is "School" or "Park", then the Parent Character cannot be randomized on that location. Rather, the Parent Character is randomized in the Secondary Backgrounds of "Home". The reason for these special cases is to try to make the story more believable, ie. trying to mimic what happens in real life. The next task of the exposition step is to add the Static Characters in the Secondary Backgrounds located in the Primary Background to the parent characters. This is because they are also considered as adult characters and may perform the same function as a "secondary" parent. However, parent characters (Mother and Father) cannot be considered as Static Characters.

The *names* and *locations* of the main character, child characters, and parent characters are then stored into memory. Also, additional states are kept track for the main character. This is the *Character Problems* that he/she currently has and the *current action* he/she is doing. The problem are used by the Story Planner for the Story Planning while the current action is used in the IS;Why;CharAction dialogue which will be discussed later.

Finally, the exposition forms *Sentence Specification Simple Sentences* that states the setting of the story. The Sentence Specification Simple Sentence is a sentence specification that is used to define a simple sentence or narrative sentence that happens in the story.

Add current Action / Event to executed Actions

Going back to the Story Executor, the next task of the Story Executor is to execute the action / event it is currently processing. It also stores it into memory called the *Action Sequence* which stores all the events that have already been executed.

Moving the main character to the proper location

The Story Planner will move the main character to the location where the event is going to take place. In doing this, the Story Planner takes note of the characters that are with the main character in the location (includes Child Characters and Adult Characters) which is later used for the *requiresMinimumCharacters* relation. Additionally, the adult characters will also be updated in order to remove those that cannot be found in the new Primary Background. The main character may also possible say goodbye to the other characters in his/her previous location depending on the need (some events do not trigger goodbyes). *Sentence Specification Simple Sentences* to represent this are specified by the Story Executor.

Finally, if the location of the main character is successfully updated, then the Story Planner forms a *Sentence Specification Simple Sentence* that the character has moved from one location to the other. Also, the Story Planner will introduce the characters in the location with the main character and create the appropriate *Sentence Specification Simple Sentences.* The Story Planner tries to avoid redundant introduction of characters in locations by keeping track of the locations where the characters have already been introduced.

Additionally, there is also a function that lets the main character move along with some additional characters specified to a certain location. When this function is called, it performs a similar function to that of updating the main character's location but this time, it also updates the current locations of the other characters specified.

Satisfying Minimum Character Requirements

```
satisfyMinimumCharacterRequirements(MinimumCharacterRequired,
dialogueSpeaker, dialogueHearer)
{

    if(MinimumCharacterRequired < characters in location)
    {
        do
        {

        Invite a Child character or Parent character

        if Child or Parent character Agrees
            Update character locations
```

```
                    break;
                else
                    Find another character

                }while(MinimumCharacterRequired < characters in location)
            }

        }
```

The next task of the Story Executor is to check if there is a dialogue triggered. If there is a dialogue triggered, it checks if there are enough characters in the location to participate. If not, it triggers an algorithm that will enable the main character to invite other characters to the location.

Depending on the *dialogueSpeaker* or *dialogueHearer* relation (whichever is not the main character), the main character tries to find a character that he/she can invite to his/her previous location in order to perform the event. The dialogue used in invitation to location is categorized under the Persuasion Dialogue which will be discussed later. He/she can invite a child character, or an adult character depending on the abstract *dialogueHearer* or *dialogueSpeaker* <ChildCharacter>, <ParentCharacter>, or either a Child or Parent character. If the abstract is <either> which means either a Dynamic Character or a Parent Character, the story planner will choose the abstract (<ChildCharacter> or <ParentCharacter>) whichever is closest to the location of the main character. However, a special case is if the event can be executed anywhere. In this case, the main character does not invite other characters, but he/she just goes to look for a close location where there may be other characters to satisfy the minimum characters needed to perform the event. Characters that are asked by the main character can agree or disagree. If the character agrees, he/she comes with the main character. If not, the main character goes again to look for another character to invite until he/she satisfies the minimum characters needed. Additionally, invited characters come with the main character when looking for other characters to invite. In going to locations, the appropriate *Sentence Specification Simple Sentences* to represent this are specified by the Story Executor.

Once the minimum number of characters needed to perform the event are done, then the Story Executor converts the abstract *DialogueHearer* or *DialogueSpeaker* to a proper character name located in the location with the main character.

Satisfaction of Event Requirements

The next task of the Story Executor is to satisfy the requirements the event has. The requirements of the event is specified in the *Requirements* column on the *Actions* table in the Knowledge Base. There are two (2) requirements that are checked: (1) NeedObject(<Object>) and (2) askDo.

```
hasRequirements(Requirements)
{

    if(hasRequirements(NeedObject))
    {
         Find <ATrans Action>

         execute <ATrans Action>

         if <ATrans Action> Type is specificInstance
         {
              Find an instance of neededobject

              if <ATrans Action> triggers Negotiation dialogue
                   Call DialogueHandler(Negotiation dialogue)
              else
                   Call DialogueHandler(Information-Seeking dialogue)

              Gain object
         }
         else if <ATrans Action> type is listOfOptions
         {
              Call DialogueHandler(Information-Seeking dialogue)

              Gain object
         }
    }
    else if(hasRequirements(askDo))
    {
         Call DialogueHandler(Information-Seeking dialogue)
    }
}
```

NeedObject requires that a certain object is needed before the event is executed. In order to gain the object, the Story Planner executes an ATrans action used in order to gain the object. There are two (2) types of ATrans: specificInstance and listOfOptions. specificInstance means that the action is used in order to gain a specific instance of an object and listOfOptions means that the action is used to select from a list of possible options given an object. For example, the object needed is ”Food”. specificInstance already finds an instance of ”food”, for example, ”hamburger” and uses it already on the dialogue (”Can I have a hamburger?”), while listOfOptions triggers a dialogue that specifically allows the receiver to choose from a list of instances of the object, ex. ”Can I have food?” -> ”I will give you food” -> ”What do you give?” -> ”I give spaghetti”, ”I give Hamburger”, ”I give water”, etc. The relations *hasGiver and hasReceiver* specifies the receiver and giver respectively. It can have the values <MainCharacter>, <ChildCharacter>, <ParentCharacter>, <StaticCharacter>, or <Child> or <AdultCharacter>.

The *hasRequestType* relation specifies the type of request the ATrans is. There are two (2) possible values for this: request and knowledge. Request means that the receiver is requesting the giver to give him/her the needed object. Knowledge on the other hand means that the receiver is requesting the giver to tell him/her where he/she can find the needed object. Request triggers a ”Can” IS dialogue, while knowledge triggers a ”Where” IS dialogue. Additionally, ATrans actions may trigger Negotiation dialogues with respect to their *NegoationReady* column in the *ATrans* table in the Knowledge Base. However, only ”specificInstance” ATrans Actions can trigger a Negotiation Dialogue. Sentence specifications to represent the ATrans are called as well. Once the object is gained, it is remembered by the Story Planner.

The next requirement is ”askDo” which specifies that the DialogueSpeaker ask ”What are we going to do?” and have the DialogueHearer say the action they are going to do ie. ”We are going to learn”. This is for aesthetic purposes and to avoid the scenario where an event just seemingly starts. Also, similar to satisfying the first events, the satisfy requirements also checks if the event triggers an ”Inquiry” dialogue, and ensures that a topic is made available for the Inquiry dialogue.

Execution of actual event

The next step the Story Executor takes is to set the agens of the current event. Depending on the minimum number of characters, the agens is either the main character or set to <All>, which means that more than 1 character is going to do the event. The agens is can currently be only the main character here, because the main character is the focus of the story, and the story events is formed based on the actions of the main character.

If there is a dialogue triggered, the Story Executor then executes the specified Dialogue it triggers providing the necessary information for each dialogue. If there is no dialogue triggered, then the story planner then just forms a *Sentence Specification Simple Sentence* of the event with the appropriate inputs using the Agens, Action, and Patiens as well as complements that are used to perform the Action (such as needed object).

Addition and Subtraction of Character Problems

The next step the Story Planner will take is to Add / Remove the character problem from the main character state by checking the *createCharProblem* and *solveCharProblem* relations of the <Action> in the Knowledge Base. It is only possible to add a problem that is related to the specified Character Problem of the user. An example is if the specified Character Problem is "Hunger", then the "Knowledge" problem cannot be created even if an event that can create-Knowledge is executed. This is to ensure that the stories do not tend to get quite long and lose its plot. Adding problems is associated with the Inciting Incident plot while removing problem is associated with the Falling Action plot. *Sentence specifications* are also formed whenever a problem is formed or solved.

Performance of special events specified in the Event

The next step of the Story Executor is to do the special perform actions of the event. The possible special perform actions of the event are (1) "Incapacitate", and (2) "Prox1(<Location>)" "Prox1(<Location>)".

Incapacitate means that the main character will be unable to move / act and will need to be approached by another character in order to help him/her. This is done by calling the "Ask feel" event in the Knowledge Base. An example action that performs "Incapacitate" is the action "Cry".

The *Prox* specifies if the action moves characters to another location after the event. This is similar to updating the location of the main character, however, it also moves the other characters in the same previous location the main character was, to the new location. The location can either be a Primary Background or a Secondary Background. The characters are then "leashed" to the main character until a character problem is solved. This means that they will follow the main character wherever he / she goes. This is because it was observed that this behaviour is fitting to the actions that have a ”Prox” special perform. The appropriate *Sentence Specifications* are also done by the Story Executor. The *Prox2* works similarly but only moves the main character to the new location.

Execution of Next Event

Finally, the final task of the Story Executor is to execute the next event of the event executed if there are any. This is done by querying the *hasNextEvent* relation of the current <Action> being executed.

4.6 Dialogue Handler Module

In order to properly create dialogues in the story, the Story Dialogue Generator also needs to have some knowledge about dialogue representation. The Dialogue representation is divided into two parts: the **Main types of dialogue**, and the **Utterances**. The main types of dialogue are definitions of the goal of a dialogue, while the utterances are used to form the main types of dialogue. Dialogues are mapped to the actions / events using the *DialogueTriggered* field. The Dialogue participants (Speaker and Hearer) are also mapped to the action / event using the *Speaker* and *Hearer* field. Below is a discussion of the main types of dialogue and Utterances used.

4.6.1 Main types of dialogue

The main types of dialogue are definitions of the goal of a dialogue. They were helped identified by reviewing the work of Walton (1992) and Walton and Macagno (2007). The Story Dialogue Generator takes note of 5 types of dialogue and tailored its purpose to the domain of the research: the Information-Seeking dialogue, the Inquiry dialogue, the Persuasion dialogue, the Negotiation dialogue, and the Deliberation dialogue. Each dialogue is concerned with a different agenda for the speakers. **Table 4.15** gives the 5 major types of dialogue used by the system.

The main types of dialogue’s purpose is to define the purpose of a dialogue and depending on its purpose, also define its structure. The 4 main types of dialogue, Information-Seeking, Inquiry, Persuasion, and Deliberation are mapped to the Actions / Events using the **triggersDialogue** relation. Additionally, the Action / Event provides the speaker and hearer of the dialogue using the **dialogueSpeaker**

Table 4.15: Main types of dialogue

Dialogue Type	Description
Information-seeking dialogue	To ask a particular question to gain information, or to request for some information
Inquiry dialogue	To discuss about some subject in order to increase knowledge about that particular subject
Persuasion dialogue	To suggest ideas to other characters, to persuade others to do a particular action or activity
Negotiation dialogue	To create an agreement with others in order to get the best out of it for oneself
Deliberation dialogue	To reach a possible course of action given a specific problem

and **dialogueHearer** relations. This is to define who is going to be the first speaker in the dialogue as well as defining the dialogue participants.

The only exception is the Negotiation dialogue, which is mapped to a different table, the **ATrans table**. The Negotiation dialogue is used if an **ATrans** action can trigger a negotiation dialogue.

The Dialogue Handler is made up of the definitions of how to execute each main type of dialogue, and also definitions of the utterances that form the types of dialogues.

Information-Seeking dialogue

The first type of Dialogue to be discussed will be the **Information-Seeking dialogue**. The purpose of the Information-Seeking dialogue is to (1) ask a simple WHType question and get an answer (2) be used for getting a topic for Inquiry Dialogue (3) be used for getting a NeedObject requirement. A special type of the IS dialogue, the askDo, is used in order to start conversation when the characters are preparing to do something. The askDo is specified as a *Requirement* in the *Actions* table. An example is "What are we going to do?" "We are going to play".When used as input for the *triggersDialogue* relation, the **Information-Seeking dialogue** has the format <*DialogueType*> ; <*WHType*> ; <*WHFocus*>. Below is a discussion of the elements that the Information-Seeking dialogue needs.

The *DialogueType* is used in order to specify that the dialogue triggered is an Information-Seeking dialogue or IS. An example format would be "IS;What;NounInstance".

The *WHType* is used to specify the type of question of the dialogue. The WHType has five (5) possible types: What, Where, Why, Can, and How.

The 'What' question is used for questions pertaining to asking more information about something. An example is "What is the time?". Additionally, 'What' questions need an additional specification, the *WHFocus*, which determines to the discourse focus of the 'What' question. It can have three (3) possible values: NounInstance, ActionInstance, and PatiensInstance. NounInstance is mainly used in Inquiry dialogues to get topics related to a certain action. For example, given the action 'learn', the NounInstance specifies the different topics that can be associated to the Action 'learn' (ex. things that can be learned). It does this using the "conceptuallyRelatedto" concept and checking the concepts returned to the "GeneralNouns" table, which contains the topics available for discussion. NounInstance specifies that the discourse focus of the question is the verb or action of the phrase. ActionInstance also specifies that the discourse focus of the question is the verb or action of the phrase. This is useful in finding an applicable patiens to a certain verb. An example would be the incomplete phrase "Mary play <patiens>". The PatiensInstance on the other hand, means that the Patiens is the discourse focus of the question. It is useful in finding more specific instances of the patiens specified. For example, consider the phrase "Mary ate food". Using PatiensInstance, the story planner will try to find related instances of the patiens "food", which may lead to the possibility of transforming the sentence to "Mary ate hamburger", with "hamburger" an instance of food.

The 'Where' question is used in order to find the location of something (ex. "Where is the ball?").

The 'Why' question is used in order to find the cause of something (ex. 'Why are you crying?'). Additionally, 'Why' questions also have <WHFocus> choices: CharAction or effectOf. CharAction is used in order to find out the cause of the action the main character is currently doing. This is done by using the *actionsequence* variable that keeps track of the actions that the main character has done and finding the last action that caused the main character to do his/her current action. The effectOf on the other hand, consults the Knowledge Base for an answer instead of the actual actions that happened in the current story.

The 'Can' question is used for requests such as 'Can you give me a ball?'.

Finally, the 'How' question is used in order to find out how to do something (ex. "How will we play?").

Below is a pseudocode of the algorithm of the Information-Seeking dialogue and the tasks it performs:

```
Information-Seeking dialogue(whType, focus)
{
     if whType is "Can"
     {
          if focus is "listOfOptions"
          {
               Call Info-request
               Call Agreement
               Call Info-request
               Call Inquiry dialogue
               Call Narrative Sentence (Gain the object chosen)
          }
          else if focus is "specificInstance"
          {
               Call Info-request
               Call Agreement
               Call Narrative Sentence (Gain Needed object)
          }
     }
     else
     {
         Call Info-request
         Call Answer
     }

}
```

The Information Seeking dialogue has two main paths that it follows. The (1) IS;Can / Request dialogues and (2) Other IS dialogues.

The First path is when the Dialogue is an IS;Can dialogue. This means that the purpose of the Information Seeking in this part is to solve the *NeedObject* requirement by trying to gain the needed object through requesting it from other characters. Additionally, the IS;Can dialogue / request for an object is split into two types: (1) listOfOptions and (2) specificInstance. The IS dialogue uses a different sequence of utterances based on each option.

For listOfOptions, the IS dialogue is formed by the utterances below in order:

1. Info-request
2. Agreement
3. Info-request

4. Inquiry dialogue
5. Narrative Sentence (Gain the object chosen)

Below is an example of an IS dialogue listOfOptions with matching utterances definition:

```
Info-request: "May I have food?" asked Dennis.
Agreement: "Ok, Dennis,"  agreed Cafeteria Lady Elise.
Info-request: "What will you give?" asked Dennis.
Inquiry: "I give fruit, salad and pizza,"  added Cafeteria Lady Elise.

Narrative Sentence: Dennis had pizza.
```

The first info-request is to ask the question of requesting the needed object. The agreement is to determine whether the dialogue hearer will agree to help the character gain the object. If the hearer disagrees, the IS dialogue ends and the main character proceeds to find another character willing to help him/her. The third info-request is used in order to ask the hearer for a list of options of the things that he/she can give. An Inquiry dialogue is then used in order to give a list of options of what the main character is asking for. Finally, given the list of options, the main character chooses an object and gains that object.

For specificInstance, the IS dialogue is formed by the utterances:

1. Info-request
2. Agreement
3. Narrative Sentence (Gain Needed object)

Below is an example of an IS dialogue specificInstance with matching utterances definition:

```
Info-request: "May You give cereal?" asked Michelle.
Agreement: "Ok, Michelle,"  agreed Cafeteria Lady Elise.

Narrative Sentence: Michelle had cereal.
```

The main difference of the specificInstance is that the character knows already the specific instance of the object that he/she wants. The first info-request is to ask the question of requesting the needed object. The agreement is to determine whether the dialogue hearer will agree to help the character gain the object. If the hearer disagrees, the IS dialogue ends and the main character proceeds to find another character willing to help him/her. However, if the hearer agrees, then he/she gives the main character the object he/she is seeking for.

The Second path, is for the other WHType questions, it also branches out to two paths: (1) The dialogue type is IS;What;NounInstance and (2) other WHType questions. The reason for this is that the *IS;What;NounInstance* is usually followed by Inquiry dialogues and needs to set a global variable *Topic* that is used by the Inquiry dialogue in creating its dialogues. The only difference between the two is that IS;What;NounInstance sets the topic, while the other WHType questions does not. They are both formed by the utterances:

1. Info-request
    (a) Answer

Below is an example of an IS dialogue that falls under the other category with matching utterances definition:

```
Info-request: "What will we learn?" asked Michelle.
Answer: "We will learn about animal," answered Teacher Lady.
```

Info-request and Answer are called only as Info-request but the Answer utterance is called inside the Info-request utterance to answer the question already.

Inquiry Dialogue

The Inquiry dialogue is used to discuss about some subject in order to increase knowledge about that particular subject. When specified as input for the *triggersDialogue* relation, it only has the format of <DialogueType>. An example format would be "Inquiry".

Inquiry dialogues are used when the characters seek to know more about the topic being discussed. Inquiry dialogues are closely associated to MTrans actions. The relations that are used by the Inquiry Dialogue in discussing more about the topic are: ***isA***, ***madeOf***, and ***partOf***. Inquiry dialogues also rely on the relation ***isGeneralNoun*** in order to properly know the possible relations the topic can be discussed about. What questions are used to retrieve information about the target object from the knowledge base.

The Inquiry dialogue is used after an Information Seeking dialogue, specifically, an IS;What;NounInstance dialogue, in order to learn more about the topic set by the IS;What;NounInstance. The utterances that form the Inquiry dialogue is a Narrative sentence followed by a set of alternating (1) Answer utterances followed by (2) Statement utterances:

1. Narrative Sentence
2. Answer
3. Statement

Below is an example of an Inquiry dialogue with matching Utterances used:

```
Narrative Sentence: Michelle, Erika, Paul and Teacher Lady learned animal.

Answer: "Wolf is an animal," answered Paul.
Statement: "Wolves can howl," said Erika.
Answer: "Dog is an animal," answered Teacher Lady.
Statement: "Dogs can bark," said Erika.
Answer: "Pig is an animal," answered Teacher Lady.
Statement: "Pigs can grunt," said Erika.
Answer: "Giraffe is an animal," answered Paul.
Statement: "Giraffes can graze," said Erika.
```

The purpose of the Narrative sentence is to introduce what the characters are going to do. For example, they are going to learn about animals ("Paul and Teacher Lady learned about animals"). The purpose of the Answer utterance is to give the answer to the standing question introduced by the IS dialogue. The purpose of the Statement utterance on the other hand, is to provide additional information that is in correlation with the *Lesson* selected by the user. More discussion regarding this will follow. The inputs of the Inquiry dialogue are speaker, receiver, agens, action, patiens, iterations, StoryPlanner sp, String lesson, String inquirytype. The notable new parameters here are the *iterations* and the *Inquiry type*. The Iterations depict the number of answers that the Inquiry dialogue will provide, while the Inquiry Type changes the form of the way the characters will discuss about the topic.

Below is a pseudocode of the steps the Inquiry dialogue takes:

```
Inquiry(<Action>, Topic)
{
     Find the MTrans Type of <Action>

     Find the possible relations that can be associated
     with the current Topic

     for i = 0, i < possibleanswers || iterationvalue, i++
     {
         Answer utterance
         Statement utterance
     }
}
```

The first step is to determine the MTrans type or type of Inquiry. The MTrans Type is important because it affects the way the characters discuss about the Topic. The Inquiry Types are retrieved by consulting the *MTrans* table. There are a total of five (5) different Inquiry types: (1) CharHear (2) CharHear2 (3) CharAction (4) CharAction2 and (5) listOfOptions.

CharHear and CharHear2 is a type of inquiry that depicts that the way that the characters will learn is through the process of discussion or conversing with each other. However, there are certain differences between the two. The Answers in CharHear are formatted in the form of <Answer> <Relation> <Topic>. No matter the MTrans action, it will follow this form. Also, CharHear is not location specific, meaning that any answer can be given and not filtered out based on location. This is because CharHear tries to simulate that the answers the characters are giving is coming from their own knowledge or from what they know. An example is given below in Italic Form.

```
MTrans action name: learn

"What does we do?", asked Paul
"We will learn.", said Teacher Lady
"What does we learn?", asked Paul
"We learn about food.", answered Teacher Lady
```

```
Paul, Michelle and Teacher Lady learned food.

"Hamburger is food.", answered Michelle
"Vegetable is food.", answered Teacher Lady
"Pancake is food.", answered Teacher Lady
```

The CharHear2 however, follows a different format to that of the CharHear. In CharHear2, it uses the MTrans action name inside the statement of a character. Also, CharHear2 filters out possible answers that are not on location. The reason for this is to simulate the scenario that the characters are engaging in conversation with what they are experiencing in the current environment. An example is given below:

```
MTrans action name: see

"What does we do?", asked Paul
"We will see.", said Zoo Keeper Roger
"What does we see?", asked Paul
"We see a animal.", answered Teacher Lady

Paul, Teacher Lady and Zoo Keeper Roger saw animal.

"I see a wolf.", answered Zoo Keeper Roger
"I see a giraffe.", answered Teacher Lady
"I see a lion.", answered Zoo Keeper Roger
```

CharAction and CharAction2 on the other hand, depict that the discussion process is performed by the characters instead of discussing about it orally. In the same way as CharHear2, it uses the MTrans action name in the creation of the narrative sentence. Additionally, it combines two sentences together separated by a "#". It creates a simple narrative sentence with the form of <Agens> <MTrans Action> # <Answer> <Relation> <Topic>. In order to combine the sentence, it uses the *CoordinatedPhrase* in the surface realizer, SimpleNLG. Another difference between the two is that CharAction2 is location specific in finding the answers while CharAction is not location specific. The reason for this is because CharAction is designed for actions such as "read" that can transfer many kinds of knowledge to a character regardless of location. The CharAction2 on the other hand is designed for actions that allow the character to learn through experiencing it in the world. An example is see. An example of CharAction and CharAction2 is given below:

```
MTrans action name: read

Erika, Michelle and Librarian Poppy learned animal.
\textit{Erika reads that goat is animal.
Erika reads that dog is animal.
Erika reads that lion is animal.}


MTrans action name: see

"What does we do?", asked Michelle
"We will see.", said Zoo Keeper Roger
"What does we see?", asked Michelle
"We see a animal.", answered Zoo Keeper Roger

Michelle and Zoo Keeper Roger saw animal.
Michelle sees that lion is animal.
Michelle sees that wolf is animal.
Michelle sees that giraffe is animal.
```

Finally, listOfOptions performs similarly to the CharAction2. It is used when the character is asking for a list of possible objects to select from. It has a different from CharAction2 and the others in which it already includes into only 1 sentence the answers returned by the Answer utterance (makes use of the NoAnswer feature of the Answer utterance). An example listOfOptions is given below:

```
"Hello, Cashier Nina,"  greeted Dennis.
"Hello, Dennis,"  greeted Cashier Nina.
"May I buy food?" asked Dennis.
"Ok, Dennis,"  agreed Cashier Nina.
"What will you sell?" asked Dennis.
"I sell fried chicken, spaghetti, egg and cereal," said Cashier Nina.
```

The next step that the Inquiry dialogue will do is to find the possible relations that they can discuss about of the Topic. The possible relations that can be used for discussion about the Topic is defined by the **isGeneralNoun(Job,<GeneralNounType>)** relation and contain the necessary information to provide the possible relations that can be discussed about it (isA, madeOf, partOf). Once the Inquiry dialogue has queried the isGeneralNouns relation to find out the possible relations, it randomly selects the possible relation that will be discussed about the Topic.

The next step by the Inquiry dialogue is to call the *Answer* utterance and provide it with the necessary information such as the Topic, the MTrans Type, and the Relation to be discussed about the topic. The Answer utterance then forms the correct sentence specifications depending on the input given to it. How exactly the Answer utterance handles this is discussed in its own section.

The next step the Inquiry dialogue takes is to call additional statements that will support the Answer to the standing question. The additional statements main purpose is to satisfy the presence of the *Lesson* that the user specified as an input. There are three (3) possible additional statements that the Inquiry dialogue will provide; one for each possible *Lesson* Noun, Verb, and Adjective.

If the Lesson selected is a Noun, then the Inquiry dialogue no longer forms additional statements to support the answer. This is because of the nature of the Inquiry dialogue in the case that it already discusses about Nouns in general.

If the Lesson selected is Verb, then the Inquiry dialogue provides additional Statements that supports the Answer utterance. Given the <Answer>, it consults the Knowledge Base for the relations *canDo* and *capableOf*. These two relations are considered because these two relations are generally associated with Verbs. The Inquiry dialogue then randomly selects which relation to use (as long as there is an existing concept found). If it chooses the *canDo* relation, it then provides an additional statement with the form of <Answer> 'can' <canDo>. An example would be "Lion can roar.". If it chooses the *capableOf* relation, then it provides an additional statement with the form <Agens> 'can' <capableOf> <Answer>. An example would be "You can chop a vegetable".

If the Lesson selected is an Adjective on the other hand, then the Inquiry dialogue consults the Knowledge Base for *propertyOf* and *Modifiers* relations in order to find applicable adjectives to the answer. Once it finds an applicable adjective to the Answer, it forms the statement with the form <Answer> is <Adjective>. An example is "Lion is brave.".

Finally, the Inquiry dialogue continues to iterate and continue discussing about the topic until there is no more to discuss or the specified iterations is reached. It is also worth mentioning that for each iteration, the Inquiry dialogue will remove the Answers that were already discussed in order to avoid them being repeated in the discussion.

Additionally, the Inquiry dialogue also triggers a function that will add that an Inquiry dialogue happened in the story regarding a particular Topic. This is going to be used by the Story Planner in forming its conclusion and summarizing the new things that the main character learned in the story.

Persuasion Dialogue

The Persuasion dialogue currently has two main purposes: (1) To have a character suggest a solution to the main character to solve his / her problem (SuggestSolution) and (2) To have the main character invite other characters to his / her current location (InviteToLocation). The SuggestSolution is used by the story planner in order to specify that another character will explicitly tell the main character a solution to his/her current character problem. This is used to make the idea that the solution came from someone, possibly more knowledgeable, than the main character itself. The InviteToLocation is used in order for the main character to persuade other characters to join him/her on a certain location in order to do something that requires 2 or more characters.When specifying the Persuasion dialogue as input to the *triggersDialogue* relation, it has the format <DialogueType> ; <PersuasionType>.

Given below is a pseudocode of the steps the Persuasion dialogue takes to create the proper dialogue. Included in the pseudocode is the paths "SuggestSolution" and "InviteToLocation".

```
Persuasion(persuasiontype, characterproblem)
{
     if persuasiontype is SuggestSolution
     {
          Call Statement utterance

          Find an Action / Event that can SOLVE the character problem

          do
          {
               Randomly select a possible action / event from
               the list of action / event returned

               Call Statement utterance
               Call Agreement utterance

          }while(Agreement == false)

          Call storyExecutor(<Action> selected)
     }
     else if persuasiontype is InviteToLocation
     {
          Inforequest utterance
          Agreement utterance

          if agreement failed
               return Failed
          else
               return Success
     }
}
```

Persuasion;SuggestSolution

The first purpose of the Persuasion dialogue is to have another character help the main character reach a possible solution to his / her problem. The suggest solution path can be triggered in two ways: first, it is automatically triggered when the main character is incapacitated. This is to create the illusion in the story that the main character cannot act on his / her own while incapacitated and needs to be helped / consoled by the other characters in the story. The second way it can be called is by being called by the Falling Action as the suggest solution path automatically depicts that it can lead to a solution to the main character's problem. This path is formed by the following Utterances.

1. **Statement utterance**
2. **Statement utterance**
3. **Agreement utterance**

Below is an example of a persuasion dialogue occuring with the Utterances used:

```
She does nothing in Living room.
She felt sad.
She cries in Living room.
She meets Mommy Emma and Daddy Will in Living room.
Mommy Emma talked to Michelle.

"Hello Michelle.",  greeted Mommy Emma
"Hello Mommy Emma.",  greeted Michelle
"Why does you cry?", asked Mommy Emma
"I cry because I do nothing.", answered Michelle
```

```
Daddy Will talked to Michelle.

"Hello Michelle.",  greeted Daddy Will
"Hello Daddy Will.",  greeted Michelle
Statement utterance: "I want to feel Happy.", said Michelle

Narrative Sentence: Daddy Will told Michelle a solution.

Statement utterance: "You will play in Living room.", said Daddy Will
Agreement utterance: "I will play Daddy Will.",  agreed Michelle
```

A discussion of the steps that the Suggest Solution path takes will be discussed here.

The first step that the Suggest Solution path takes is to have the main character (which has the problem) state to the dialogue hearer his / her problem in order to create the illusion that the dialogue hearer has enough information in order to suggest a solution to the main character depending on his / her problem.

The second step that the Suggest Solution path takes is to select the following actions that can solve the main character's problem. This is done by consulting the *Actions* table in the Knowledge Base in a similar way to that of the Falling Action. Also, similar to the Falling Action, it filters the possible Actions that are returned using the priority functionalities in the Falling Action.

The next step that the Suggest Solution path will take is to select the a random action solution from the possible Actions. It also selects the location to which the action is going to take place. The Suggest Solution will then have the character (suggester) to issue a statement directed to the main character inclining him / her to do this action to solve his / her problem.

The next step that the Suggest Solution path will take is to have the main character agree or disagree with the proposal of the character. If the main character disagrees, the suggester will then find another possible solution that can be persuaded to the main character. Also, it eliminates the action solutions that have already been suggested. This is to give way to other possible solutions to be selected. However, if there are no more possible solutions, the suggester will repeat his / her earlier propositions to the main character until the main character finally agrees.

Once the main character agrees to the proposition of the user, he / she then follows to do the action stated by the suggester. The Suggest Solution does this by calling the Story Executor in the Story Planner module. Additionally, it specifies that there is a specific location to which the action should be executed.

Persuasion;InviteToLocation

The other path that the Persuasion dialogue can take is the *InviteToLocation* path. This path is used by the Story Planner when the main character needs to invite other characters to a location where he / she needs to perform an action. The InviteToLocation path is called by the Story Planner when it finds out that the *satisfy minimum characters needed* fails. The InviteToLocation path is formed by the following Utterances followed by an example Story excerpt pf the InviteToLocation dialogue with matching Utterances used:

1. **Inforequest utterance**
2. **Agreement utterance**

An example InviteToLocation is given below:

```
"Hello Erika.",  greeted Michelle
"Hello Michelle.",  greeted Erika
Info-request: "Can You come with me to Living room?", asked Michelle
Agreement: "I will come with you Michelle.",  agreed Erika
```

The first step that the InviteToLocation will take is to have the main character request to the character if he / she can come to a location to do a certain action. The important things that are passed to the inforequest are the action to be done, the location to go to, a parameter specifying it is a ”Can” question, and also a parameter specifying that there should be no answer given (as the agreement utterance will be called separately). The form the inforequest will take is in the form of ”Can <Invitee> 'come' 'with <main character>' to <location>?”. An example would be ”Can Paul come with me to Playground?”.

The next step that the InviteToLocation will do is to have the invitee agree or disagree to come with the main character. If the invitee disagrees, the InviteToLocation function returns <Failed> to the Story Planner and the Story Planner proceeds to have the main character go to other locations to invite other characters. However, if the invitee agrees, the InviteToLocation function returns <Agree> to the Story Planner and the Story Planner then proceeds to have the invitee go with the main character to the location specified. The Agreement utterance in this case has the form <Invitee> will come with <Main Character pronoun> <Main Character Name>.

Deliberation dialogue

The Deliberation dialogue is used to reach a possible course of action given a specific problem. When specifying the Deliberation dialogue as input for the *triggersDialogue* relation, it only has the format of <DialogueType> An example format would be ”Deliberation”. Deliberation dialogues are used to determine *how* to perform an action, thus, the related concept that can be applied can be either a noun or a verb. Consider the action play. When deliberating how to play, the related concepts can be either basketball which is a noun, or ride which is a verb. Since basketball is already a noun, a complete sentence can already be formed, i.e., We played basketball. However, if the related concept is a verb, then the resulting sentence will be We ride instead. A target object must also be selected to complete the sentence, e.g., bicycle to form the sentence We rode a bicycle.

The Deliberation dialogue is formed by the Utterances followed by an example Deliberation dialogue with matching Utterances used:

1. **Info-request utterance**
2. **Answer utterances**
3. **Statement utterance**
4. **Agreement utterance**
5. **Narrative Sentences**
6. Depending on the Lesson, the Deliberation dialogue calls simple sentences that correlate to the Lesson.

```
Info-request: "What will we play?" asked Dennis.
Answer: "We will play tag," answered Michelle.
Answer: "We will play hide and seek," answered Michelle.
Answer: "We will play basketball," answered Michelle.
Answer: "We will play rope," answered Michelle.
Answer: "We will play pet," answered Michelle.
```

```
Narrative Sentence: Dennis chose to play basketball.

Statement: "We will play basketball," said Dennis.
Agreement: "Ok, Dennis,"  agreed Michelle.

Narrative Sentence: Dennis and Michelle played basketball.
Narrative Sentence: Dennis loudly caught a color green ball.
Narrative Sentence: Michelle smoothly dribbled a color green ball.
Narrative Sentence: She highly shot a color green ball.
```

Below are the steps taken by the Deliberation dialogue in order to generate the dialogue between the characters:

```
Deliberation(<Action>, Lesson)
{
     Call Info-request utterance

     if Lesson is Verb
     {
         use howTo relation
         for i = 0, i < iterations, i++
             Answer utterance
     }
     else if Lesson is Noun
     {
         use conceptuallyRelatedTo relation
         for i = 0, i < iterations, i++
             Answer utterance

         if no conceptuallyRelatedTo relation
         {
              use howTo relation
              for i = 0, i < iterations, i++
                  Answer utterance
         }
     }
     else if Lesson is Adjective
     {
         for i = 0, i < iterations, i++
             Answer utterance
```

```
        if no conceptuallyRelatedTo relation
        {
             use howTo relation
             for i = 0, i < iterations, i++
                 Answer utterance
        }
    }

    do
    {
         Create Narrative Sentence
         Call Statement utterance
         Call Agreement utterance
    }while agreement == failed

    Have the characters do the <Action>

    Add additional narrative sentences depending on the lesson
}
```

The first step that the Deliberation dialogues takes is to call an Info-request utterance that has one of the dialogue participants state how they are going to do a specific action.

The second step that the Deliberation dialogue takes is to call the Answer utterance in order to find the possible ways to do the action specified. In finding the possible ways to do the action, there are two possible paths that the Deliberation dialogue will take: (1) the **How route** and the (2) **What route**. The path to take depends on the *Lesson* specified by the user. The 'How' route is taken when the Lesson is 'Verb'. This is to ensure that more verbs are going to be displayed because the 'How' route contains mainly verbs because of the specified relation it uses 'howTo' (which is composed of verbs). The 'What' route on the other hand is taken when the Lesson is 'Noun' or 'Adjective'. The 'What' route finds concepts on how to do the action based on the relation *conceptuallyRelatedTo*. An example is the 'play' conceptuallyRelatedTo 'basketball'. The 'conceptuallyRelatedTo' relation is used in this instance because it primarily contains the concepts that are considered 'Nouns'. The Adjective lesson also uses the the 'What' route in order to properly display the most adjectives. However, there are instances when the 'What' relation fails as there are no *conceptuallyRelatedTo* relations with regards to the action. If this is the case, then the Noun and Adjective lessons use the 'How' route. For the case of the Noun lesson, the 'How' route can still satisfy the requirements of the lesson in displaying the Nouns. For the case of the Adjective lesson, additional specifications will be done in order to still display Adjectives in the story.

The 'How' route tries to find concepts that are related on how to do the action. In this path, it specifies to the Answer utterance that it is looking for a *howTo* relation with regards to the action. An example relation is 'play' has a howTo connection with 'ride'. However, the information needed is still incomplete. The answer 'ride' for example, is a verb and seems awkward if used as is. What is needed is a patiens to complete the answer. What happens is in the Answer utterance, an additional process is done in which a patiens is found by consulting the *capableOf* relation and finding the concepts that the answer verb is related to. For example, the 'ride' has a capableOf relation to 'bike'. Additionally, the Answer utterance filters out returned concepts here that cannot be located in the current location of the main character. This is to avoid incoherencies in the story such as 'I rode a bike in the living room'. If there are no applicable patiens to the verb, the answer utterance is cancelled and another one starts to try to find a new answer verb. The format of the Answer utterance returns is <Agens> <Answer Verb> <Patiens>. In this case, what happens is something like "Paul rode a bike". Similar to the Inquiry dialogue, the Deliberation dialogue does not choose answers that have been already stated.

The 'What' route tries to find concepts that are related to the action using the *conceptuallyRelatedTo* relation. It calls the Answer utterance and specifies that the relation to find is *conceptuallyRelatedTo*. An example answer here is 'basketball', where play is conceptuallyRelatedTo basketball. The answer utterance then takes the form of <Agens> <Action> <Answer>. In this example, "Paul played basketball".

The next step is to have the characters decide on what they are going to do. Since they have already stated the possible ways to do the action using the Answer utterance, they need to decide on what to do given that list of options. The Deliberation dialogue does this by randomly selecting a chosen action from the list of options. The Deliberation dialogue then forms a simple sentence indicating that a character wants to do the chosen action, and having the character state his / her choice using the *Statement* utterance. Another character must then agree using the *Agreement* utterance. If however, the character disagrees, then another answer verb is chosen from the list of options. If the list of options is exhausted, then previous propositions may be selected again. Once an answer verb has been agreed upon, the Deliberation dialogue has the characters do the action they chose. Also, since the "How" route and the "What" route has different formats, the appropriate format will be used by the Deliberation dialogue in the utterances.

Finally, depending on the lesson, additional statements are added by the Deliberation dialogue. First, if the lesson is Verbs, it does not do anything anymore because the Deliberation dialogue inherently is already concerned with Verbs.

However, if the lesson is Noun and the "What" route is taken, then additional statements are added by the Deliberation dialogue. The Deliberation dialogue finds concepts that are *conceptuallyRelatedTo* the chosen answer. An example is given the phrase "Paul played toys", where "toys" is the answer chosen by the "What" route that the characters will do. The Deliberation dialogue finds conceptuallyRelatedTo instances of "toys" such as "robot", "marbles", "toy cars", etc. The How route triggers the same additional statements as the What Route.

If the lesson is Adjective and the "What" route is taken, the Deliberation dialogue adds additional statements in order to display adjectives in the story. The first thing it does is to find a concept related to the chosen answer the will do (ex. basketball), with the relation *performedWith*. In this example, "basketball" has a *performedWith* relation with "ball". It then keeps track of this chosen concept. After that, it finds concepts related to the chosen answer with the relation *howTo*. In this example, "basketball" has a howTo relation with the concepts 'shoot', 'dribble', 'pass'. The Deliberation dialogue also keeps track of this concept. After that, it finds applicable adjectives to both the performedWith object and the howTo concept. It does this by consulting the *propertyOf* and the *Modifiers* table. Once it has found applicable adjectives to the concepts, it then combines it to create a simple sentence with the form <Agens> <howTo Modifier> <howTo Answer> <performedWith Modifier> <performedWith object>. An example is "Paul quickly passed the dirty ball".

If the lesson is Adjective and the "How" route is taken, it does the same process similar to the "What" route, but instead of applying it as additional statements, it is already done in the *Answer utterances*.

Negotiation dialogue

Finally, the Negotiation dialogue is used to create an agreement with others in order to get the best out of it for oneself. The Negotiation dialogue is a dialogue used mainly to make the story more interesting. For **Negotiation dialogues**, the concern involves identifying the possible action or event that the negotiator would propose to the character who has a need (specifically, the requirement *NeedObject*). Negotiation dialogues are closely associated with ***ATrans*** actions. If an ATrans action has a *canTriggerNegotiation* value of 1, the negotiation dialogue will be triggered.

The Negotiation dialogue has a character, the negotiator, negotiate with the main character, the negotiatee, to do something first with him / her before he / she will help in the negotiatee to receive the object he / she needs. There are two paths of Negotiation dialogues: (1) **Knowledge** and (2) **Request**. The Knowledge path is used to handle ATrans actions that request for knowledge of the location of the NeededObject, while the Request path is used to handle ATrans actions that request for the NeededObject itself. Below is a pseudocode of the Negotiation dialogue:

```
Negotiation dialogue(NegotiatorType, NegotiationType)
{

     Find Action / Event that can be used as a proposition
     for the negotiation by the Negotiator

     if NegotiationType is Knowledge
     {
          Call Inforequest utterance
     }
     else if NegotiationType is Request
     {
     }

     do
     {
          Select a random Action / Event from the list of
          possible propositions

          Call Statement utterance
          Call Agreement utterance

          if agreement == true
          {
               Perform the proposition

               if NegotiationType is Knowledge
               {
                    Answer utterance
               }
               else if NegotiationType is Request
               {
                    Narrative Sentence
               }
          }

     } while agreement == failed

}
```

Negotiation;Knowledge

The first to be discussed is the Negotiation;Knowledge dialogue. This negotiation is used to handle ATrans actions that request for knowledge of the location of the NeededObject. The Negotiation;Knowledge dialogue is formed by the Utterances:

1. **Inforequest**
2. **Statement**
3. **Agreement**
4. **NarrativeSentences**
5. **Answer**

Below is an example of Negotiation;Knowledge dialogue with matching Utterances:

```
"Hello Paul.",  greeted Michelle
"Hello Michelle.",  greeted Paul
Info-request: "Where is food located in?", asked Michelle
Statement: "You will play with me first in Living room.", said Paul
Agreement: "I will play with you Paul.",  agreed Michelle

Michelle and Paul ran to Living room.
Narrative Sentence: Michelle and Paul played.
Narrative Sentence: They rode the bike.

Answer: "Food is located in a Cafeteria.", answered Michelle
"Thank you Michelle.",  said Paul
"You are welcome Paul.",  said Michelle
```

The first thing the Negotiation dialogue will do is to get the abstract representation of the Negotiator. This is explicitly stated as a parameter when calling the Negotiator dialogue. The reason that the abstract representation of the Negotiator is needed is because actions are separated into two categories: (1) Child Negotiables and (2) Parent Negotiables. Child Negotiables are actions that can benefit the Child characters in some way while Parent Negotiables are actions that can benefit Parent characters. One important thing to note is that Static Characters cannot engage in Negotiation dialogues with the main character.

The next step that the Negotiation dialogue does is to find the possible Actions that can be used as propositions by the Negotiator. It does this by consulting the *Actions* table and checking the *isNegotiationEvent* of each action. It matches the Abstract Negotiator type with the specified isNegotiationEvent value in order to select either Child Negotiables or Parent Negotiables. It also performs location filtering on the returned Actions, similar to that of the Falling Action.

The next step that the Negotiation dialogue will do is to select a random propositions from the list of propositions returned from the previous step. Similar to the Story Executor, the location the action will take place will be chosen and closer backgrounds will be prioritized.

The next step that the Negotiation dialogue will do is to have the Negotiator state his / her proposition to the Negotiatee using a *Statement* utterance.

The Negotiation dialogue will then have the negotiatee either Agree or Disagree with the proposition of the Negotiator. If the case is that the Negotiatee disagrees, the Negotiated dialogue will find another possible proposition aside from those already stated and have the Negotiator state it again to the Negotiatee. This continues on until a proposition has been agreed upon. If the list of unique propositions have been exhausted, the Negotiator will repeat his / her earlier propositions until the Negotiatee agrees.

Once a proposition has been agreed upon, the Negotiation dialogue will then have the Negotiator and the Negotiatee do the action specified. However, unlike in the Persuasion dialogue, the Story Executor is not called. The reason for this is because it is more similar to the structure of the Deliberation dialogue in there is a "How" path and a "What" path. First, it checks if there is a *howTo* relation related to the *ActionName* of the selected proposition. If there is it takes the 'How' path. However, if there is none, it takes the "What" path.

In the case of the "What" path, the Negotiation dialogue consults the Knowledge base for possible concepts *conceptuallyRelatedTo* to the action proposition. An example relation is 'play' conceptuallyRelatedTo 'toys'. The Negotiation dialogue then has the negotiator and the negotiatee do the complete action a number of iterations. First it selects the Patiens from the conceptuallyRelatedTo concepts returned. Then it forms a simple sentence with the format <Negotiatee> # <Negotiator> <Proposition Action> <conceptuallyRelatedTo chosen concept>. An example is "Paul and Mary played basketball". It also takes note of the *Lesson* specified by the user. However, the "What" path already inherently satisfies the Noun lesson, and the Verb lesson and additional processing is not done any more. However, for the Adjective lesson, the Negotiation dialogue consults the Knowledge Base for possible adjectives with regards to the conceptuallyRelatedto concept selected using the *propertyOf* table and the *Modifiers* table.

In the case of the "How" path, it keeps track of the "howTo" verbs that are related to the proposition action. An example relation of this is the action "cook" has a howTo relation with "chop". It then iterates through the possible howTo verbs and performs the howTo verbs. In order to do this, the first thing it does is to find a concept related to the howTo verb using the relation *capableOf.* The capableOf concept's purpose is to complete the howTo verb by giving it a concept on which it will be performed. An example is the howTo verb "chop" has a capableOf to "vegetable". Also, in finding the capableOf concept, the Negotiation dialogue applies a location specific filtering and returns only the concepts that can be found in the current location of the story. An example is this: the proposition action is "clean", the howTo verb is "chop", and the capableOf found is "vegetable" and "beef". Therefore, the sentence may become "Paul chop vegetable".

Additional processing is done in order to satisfy the *Lesson* specified by the user. If the Lesson is Noun, the Negotiation dialogue tries to include additional concepts in the dialogue that will help display more nouns in the story. However, it only does this if there is an *capableOf* concept related to the howTo verb. If there is an capableOf concept and the lesson is Noun, then it tries to finds concepts that are related to the howTo verb with the relation *canDo*. The purpose of this is to find an *Instrument* that can be used to perform the howTo verb. Taking the previous example, The Negotiation dialogue then creates a simple sentence with the form <Negotiatee> # <Negotiator> <howTo verb> <capableOf concept> <canDo concept>. An example is "Paul and Mary brushed the windows with a brush".

However, if the lesson is a Verb and the path taken is "How", no additional processing is done anymore because it inherently displays varying verbs. (However, if the path taken is "What", then the Negotiation dialogue will find a *canDo* or *capableOf* relation to the conceptuallyRelatedTo concept and issue a statement with regards to that)

Finally, if the lesson is Adjective, then the Negotiation dialogue finds possible adjectives to both the Patiens and the howTo verb using the *propertyOf* and *Modifiers* table. It then creates a simple sentence with the form <Negotiatee> # <Negotiator> <howTo adjective> <howTo verb> <capableOf adjective> <capableOf concept> <canDo concept>. An example is ”Paul and Mary quickly brushed the dirty windows with a brush”.

Finally, after the Negotiation dialogue has the Negotiator and Negotiatee do the proposed action, the Negotiation dialogue has the Negotiator answer the Negotiatee using an *Answer* utterance the location of the needed object.

Negotiation;Request

The second to be discussed is the Negotiation;Request dialogue. This negotiation is used to handle ATrans actions that request for the NeededObject itself. The Negotiation;Knowledge dialogue is formed by the Utterances:

1. **Inforequest**
2. **Agreement**
3. **Statement**
4. **Agreement**
5. **Narrative Sentences**

Below is an example of a Negotiation;Request dialogue with matching Utterances used:

```
Info-request: "Can You give me fried chicken?", asked Erika
Agreement: "I will give fried chicken Erika.", Daddy John
Statement: "You will cook with me first in Kitchen.", said Daddy John
Agreement: "I will cook with you Daddy John.", Erika

Erika and Daddy John went to Kitchen.
Narrative Sentence: Erika and Daddy John cooked.
Narrative Sentence: Erika swiftly poured good dressing.
Narrative Sentence: She greatly chopped long onion.
Narrative Sentence: She swiftly peeled great knife.
Narrative Sentence: Daddy John gave Erika fried chicken.
```

```
"Thank you Erika.",  said Daddy John
"You are welcome Daddy John.",  said Erika
```

The Request path follows a similar path to the Knowledge path. The only difference is with regards to what the character is asking. In the case of the Request path, instead of the location of the object is asked, the main character asks for the object itself. And instead of the negotiator answering with the location of the needed object, he / she gives the main character the needed object. Below is the path the Request path takes:

Table 4.16: Utterances

Utterance Name	Definition
Statement	A claim made by the speaker
Info-request	Any utterance that creates an obligation for the hearer to provide information
Conventionals	Represent greetings, farewells, thanking and responding to thanks
Agreement	Accept or reject a proposal
Answer	Utterances complying with an info-request action

4.6.2 Utterances algorithm and specifications

Each type of dialogue is formed by a set of utterances with varying arrangement depending on the purpose of the dialogue. To define the utterances, the Story Dialogue Generator has adapted some elements from the DAMSL annotation scheme (Core and Allen, 1997), as presented in **Table 4.16**. Table 4.16 gives a brief overview of the utterances used.

Additionally, Utterances serve as the main medium that allows the Story Planner and Dialogue Handler to create Sentence Specifications needed to define a sentence in the story. This is because the Utterances contain the necessary processing information to properly form the Sentence Specifications given the way they are used.

Statement Utterance

The first utterance is the **Statement**. The statement is a claim made by the speaker. An example statement is ""I am hungry," said Dennis". The statement is basically a simple sentence spoken to by a character. The needed inputs by the Statement is the *Dialogue speaker, Dialogue hearer, agens, action, patiens, instrument used to perform the statement, complements, location statement will take place, a modifier to the location, the tense of the statement, if the subject is pluralized, if the object is pluralized, if the statement is a repeated suggestion, and the verb form of the verb.* Given the inputs, the Statement utterance forms the sentence specification and adds an additional assignment to the sentence specification stating that it is a Dialogue.

Simple Sentence or Narrative Sentence

Although this is technically not an utterance, it performs a similar function to that of the Statement Utterance minus being a dialogue. An example narrative sentence is "The day is sunny". This is used in order to generate 3rd person view narratives of the events that happen in the story. It also needs similar parameters to that of the Statement Utterance minus the Dialogue Speaker and Dialogue Hearer.

Info-request Utterance

The second utterance to be discussed will be the **Info-request** utterance. The Info-request utterance is used to form questions spoken to by a speaker. An example info-request is ""What are we going to learn?," asked Paul". It takes into input the *Dialogue speaker, Dialogue hearer, agens, action, patiens, primaryDialogueType, WHType, WHFocus, WhatTypeRelation, NoAnswerSpec, complements, and tense*. The info-request is concerned with several tasks. The info-request utterance provides the Sentence Specification elements: Agens, Action, Patiens, Interrogative, DialogueType, Complement, Dialogue, Hearer, Speaker, and Tense. For these specification elements, the only things changed during processing of the info-request is the Action, Interrogative, DialogueType, and Tense. Below is a pseudocode that displays the several tasks it performs in order:

```
Info-request(Dialogue speaker, Dialogue hearer, agens, action,
patiens, primaryDialogueType, WHType, WHFocus, WhatTypeRelation,
NoAnswerSpec, complements, and tense)
{

     Create Sentence Element draft using Agens, Patiens, Complement,
     Dialogue, Hearer, and Speaker

     if WHType is "What"
     {
           Set Discourse Focus
           Add Action, Interrogative, DialogueType, and Tense elements
     }
     else if WHType is "Where"
     {
           Set Discourse Focus
           Add Action, Interrogative, DialogueType, and Tense elements
```

```
        }
        else if WHType is "Why"
        {
              Set Discourse Focus
              Add Action, Interrogative, DialogueType, and Tense elements
        }
        else if WHType is "Can"
        {
              Set Discourse Focus
              Add Action, Interrogative, DialogueType, and Tense elements
        }
        else if WHType is "How"
        {
              Set Discourse Focus
              Add Action, Interrogative, DialogueType, and Tense elements
        }

        if NoAnswer is true
             return nothing
        else
             return call Answer
}
```

The first task of the Info-request utterance is using the WHType, WHFocus, and WhatTypeRelation, it forms the sentence specifications for the type of question to be created. The WHType has five (5) possible types: What, Where, Why, Can, and How.

The 'What' question is used for questions pertaining to asking more information about something. An example is "What is the time?". Additionally, 'What' questions need an additional specification, the *WHFocus*, which determines to the discourse focus of the 'What' question and the question regarding the focus. It can have three (3) possible values: *Specific, NounInstance, ActionInstance, and PatiensInstance*. 'What' questions set the interrogative as a 'What' question.

Specific is a type of 'What' question that asks a specific question regarding the Patiens. Three (3) possible relations can be chosen to pertain to the question being asked: isA relation, madeOf relation, and partOf relation. An example question is: "What is a ball?". The sentence specifications changed by this question is the 'Action' element, which contains the relation that was asked (ex. "is a"). The discourse focus of the 'What' question is automatically the patiens specified.

NounInstance specifies that the discourse focus of the question is the verb or action of the phrase. This is useful in finding an applicable patiens to a certain verb. An example would be the incomplete phrase "Mary play <Patiens>". When NounInstance is considered, the question will be formed like "What did Mary play?". Since it is a NounInstance, the story planner will find related instances to the verb 'play', which may eventually return something like "Mary played basketball". NounInstance only finds related patienses that can be considered as a Topic for an Inquiry dialogue. The

ActionInstance is also similar to the NounInstance as it uses the verb or action as the discourse focus. However, it is not limited to selecting only those that are Topics.

The PatiensInstance on the other hand, means that the Patiens is the discourse focus of the question. It is useful in finding more specific instances of the patiens specified. For example, consider the phrase "Mary ate food". Using PatiensInstance, the story planner will try to find related instances of the patiens "food", which may lead to the possibility of transforming the sentence to "Mary ate hamburger", with "hamburger" an instance of food. For these type of 'What' questions, the 'Action' element is set to the action provided to the inforequest function.

The 'Where' question is used in order to find the location of something (ex. "Where is the ball?"). The 'Where' question functions similarly to the 'What;Specific' question, however, it automatically sets its discourse focus to the Agens, and the relation to be found to the Agens as 'locationOf'. For the sentence specification, the things that are changed are the 'Action' element which is set to the relation used (location of), and the 'Interrogative' element which is set to 'where'. Also, it automatically sets the tense to "present".

The 'Why' question is used in order to find the cause of something (ex. 'Why are you crying?'). The 'Why' question can have two paths: (1) A general question that asks about the cause of a world phenomenon or (2) A question regarding to the cause of the current main character's action. CharAction is used in order to find out the cause of the action the main character is currently doing. This is done by using the *actionsequence* variable that keeps track of the actions that the main character has done and finding the last action that caused the main character to do his/her current action. The effectOf on the other hand, consults the Knowledge Base for an answer instead of the actual actions that happened in the current story. For the sentence specification, the things that are changed are the 'Action' element which is set to the action specified, and the 'Interrogative' element which is set to 'why'. Finally, it also automatically sets the tense to "present".

The 'Can' question is used for requests such as 'Can you give me a ball?'. For the sentence specification, the things that are changed is the 'DialogueType' element which is set to 'can'. The 'Can' question uses a different element to specify it is a question type because the Surface Realizer performs special processing on it to change it to a 'Can' question.

Finally, the 'How' question is used in order to find out how to do something (ex. "How will we play?"). For the sentence specification, the things that are changed are the 'Action' element which is set to the action specified, the 'Interrogative' element which is set to 'how', and finally the 'Tense' is set to 'future'. The reason for setting the tense automatically to future is because 'how' question pertains how to do an action that is still to be done.

The next task of the Info-request utterance is to either call the Answer utterance and return the answer to the question, or to return nothing (defined by the *NoAnswerSpec*). The reason for this is that some dialogues only needs 1 instance of a question and 1 instance of an answer, while other dialogues need only 1 instance of a question but multiple instances of the Answer utterance depending on the purpose of the dialogue. This may be seen in the latter parts in the discussion of the Dialogues.

Answer Utterance

The next utterance to be discussed will be the **Answer** utterance. The answer utterance is used in order to answer a standing question in the story. The answer utterance may be called either from inside the info-request utterance, or can be called on its own depending on the need of the current dialogue. The main reason for the Answer utterance being called outside of the info-request is that it is called multiple times and there are specific functions that it is used for. The Answer utterance provides the Sentence Specification elements: *Agens, Action, Patiens, AnswerType, Answer, Dialogue, hearer, speaker, and Tense*. For these specification elements, the only things changed during processing of the info-request is the Agens, Patiens, AnswerType, and Answer. The Action is always set to the default action passed to the Answer utterance, the Dialogue is always set to true, and the hearer and speaker are always set to the specified speaker and hearer. Additionally, it can be that the Answer utterance does not form any sentence specification at all. Rather, it just returns an answer to the dialogue that called it which will be later used to perform other Sentence Specifications.

The Answer utterance performs many tasks in order:

```
Answer(speaker, receiver, agens, action, patiens, questionrelation,
lessonType, dialogueType, tense, boolean noanswer)
{

     Find possible Answers depending on questionrelation
     or dialogueType

     Filter the possible Answers using locationOf relation and
     current background of <MainCharacter>

     Select a singe answer from the list of Answers randomly

     if DialogueType is Inquiry
          Set Agens element to Answer
     else
          Set Agens element to default Agens specified

     if DialogueType is IS;Why
     {
          Do not set Patiens
          Set Answer element to questionrelation
     }
     else if DialogueType is Inquiry
          Set Patiens element to default Patiens
     else if DialogueType is Inquiry3 or Inquiry 4
          Set Patiens element to Answer (if madeOf only)
          or default Patiens
     else if DialogueType is Deliberation;How
          Set Patiens element to How & Noun object +
          Chosen Adjective or Noun Object
     else if DialogueType is Deliberation;What
          Set Patiens element to What & Answer
     else
          Set Patiens element to Specifier + Answer

     Set remaining Sentence Specification elements (Action, Dialogue,
     hearer, speaker, and Tense)

}
```

The first task of the Answer utterance is to find the list of possible answers using the question that has been passed to it, as well as the type of dialogue

Table 4.17: Dialogue types and discourse focus used

Dialogue Type	Discourse Focus
IS;Why	Action
How	Action
Action Instance	Action
Noun Instance	Action
Inquiry	Patiens
Deliberation	Action
Where	Agens
Otherwise	Patiens

that called it. The question to be passed to it usually has the format of a binary relation in the database (ex. isA, partOf, conceptuallyRelatedTo, etc.). However, there are special cases. For example, if the dialogue that called the Answer utterance is an IS;Why;CharAction, then the answer utterance does not consult the Knowledge Base. Because of the nature of IS;Why;CharAction, which means it is a question asking the main character the cause of his/her current action, the Answer utterance consults the *ActionSequence* stored by the Story Planner. The *Action Sequence* contains all the current executed events in the story.

For the type of dialogue, depending on the type of dialogue specified, the discourse focus of the question changes to either the Agens, the Patiens, or the Action. An example is given the phrase: "Dog is a Animal" and then it was changed to a question, depending on the type of dialogue there are two possible answers that the 'isA' relation can be focused upon: either using 'Dog' as an input and returning the answer as 'Animal' or either using 'Animal' as input and using 'Living thing' as the answer. Similar to the example above, the "Action" is also used by some dialogue types as the discourse focus. An example is given the phrase "We play" and transformed into a "How" question. It then selects different ways to "play" which is the Action specified. **Table 4.17** is a list of the Discourse focus used by each dialogue types:

The next task of the Answer utterance is to filter the possible answers that may be given. This uses the *locationOf* table in the Knowledge Base. However, it can be that the answers may not need to be filtered at all. This is in the case of actions that explicitly state to not filter the possible answers that can be given. An example is the MTrans Action *CharHear* and *CharAction* or the TopicGetters that is specified not to be *NotLocationSpecific*.

The next task of the Answer utterance is to find a possible answer from the list of possible answers. It does this through randomization.

The next task of the Answer utterance is to create the correct sentence specifi-

Table 4.18: Dialogue types and Sentence Specifications Agens

Dialogue Type	Agens
Inquiry	Answer to the question
Inquiry2	Default agens
Inquiry3	Default agens
Inquiry4	Default agens
Otherwise	Default agens

Table 4.19: Dialogue types and Sentence Specifications Patiens

Dialogue Type	Patiens
IS;Why	No patiens specified
Inquiry	Default patiens
Inquiry3	Answer (if madeOf only) or default Patiens
Inquiry4	Answer (if madeOf only) or default Patiens
Deliberation;How	Noun object + Chosen Adjective or Noun Object
Deliberation;What	Answer
Otherwise	Specifier + Answer

cation depending on the dialogue type. The reason for this is that some dialogues require different sentence specifications. **Tables 4.18**, **4.19**, and **4.20** is the list of the changed Sentence Specification parameters depending on the dialogue type.

The next task of the Answer utterance is to perform additional processing depending on the type of dialogue. The dialogues that have additional processing are: the "Where" question, Inquiry dialogues, Deliberation dialogue, if the dialogue is not an "IS;Why" dialogue, an Inquiry dialogue, and an Deliberation dialogue, Inquiry 3 and Inquiry 4,

Because the answer returned in the where question can consist of many locations (SecondaryBackground1 ; SecondaryBackground2 ; SecondaryBackgroundn), it already randomly selects on a specific location to return.

In the Inquiry dialogue, the selected Answer is removed from the list of possible answers so that when the Inquiry dialogue calls the Answer utterance again, it does not select the said answers anymore.

Table 4.20: Dialogue types and Additional Information

Dialogue Type	AnswerType	Answer
IS;Why	effectOf	Answer to the question

For the Deliberation dialogue, it also performs addition processing similar to that of the Inquiry dialogue where it eliminates choices in order to not select those again once the Deliberation dialogue calls the Answer utterance again. Additionally, because of the returned answers of the Deliberation;How dialogues, where the *howTo* relation is used with regards to the action, it performs additional processing. In order to explain this, an example will be given. For an example, the action 'play' has a howTo connection with 'ride'. However, the information needed is still incomplete. The answer 'ride' for example, is a verb and seems awkward if used as is (Paul rides). What is needed is a patiens to complete the answer. What happens is an additional process is done in which a patiens is found by consulting the *capableOf* relation and finding the concepts that the answer verb is related to. For example, the 'ride' has a capableOf relation to 'bike'. Additionally, if the lesson type is a verb, it finds an adjective to describe the Action and the Patiens.

For the "IS;How" dialogue, it follows the similar format of the Deliberation;How dialogue.

If the dialogue is not an "IS;Why" dialogue, an Inquiry dialogue, and an Deliberation dialogue, additional specifier special characters are appended to the Patiens depending on the question relation passed to it (ex. isA, madeOf, partOf) or if it is a topic to be discussed by an Inquiry dialogue.

The Inquiry3 and Inquiry4 types of Inquiries also need additional processing because of their format in which two sentences has been combined. An example is sentence 1 is "Paul sees" and Sentence 2 is "Giraffe is animal". The combined sentence will then be "Paul sees that Giraffe is animal". It specifies additional sentence specifications particularly by creating another sentence specification by using an '' sign to indicate a new start of the sentence to be combined.

Finally, the last task of the Answer utterance is to return the chosen answer back to the function that called it (can be either the info-request or a dialogue). The reason for this is that there may be additional processing done by the dialogue to correct form the story.

Agreement Utterance

The next utterance to be discussed is the **Agreement** utterance. The agreement utterance is used on dialogues that contains some kind of request or negotiation from a character speaking. It allows the characters to agree or reject a proposal. In order to determine if a character agrees or rejects the proposal, the system randomizes between the two. There is an 80% chance that the proposal is agreed to and a 20% chance that the proposal is rejected. The agreement utterance then returns if the character agreed or rejected the proposal. Agreement words such as "Ok" are used when the character agrees with the proposal. If the character disagrees however, the proposal is *negated* by defining it in the Sentence Specifications. For example, the proposal is "You will play with me first in playground," said Erika. The disagreement then is "I will not play, Erika," said Michelle.

Greetings and Closings

The next utterances are the **Greetings and Closings**. Greeting utterances are divided into two parts: Initial Greeting and Greeting Response. In the Initial Greeting, the system checks if the two Dialogue participants have already greeted each other throughout the story. If it finds out that they have already greeted each other, then the system does nothing any more. However, if it founds out that the participants have not greeted each other yet, then the system initializes a Greeting between the participants in the dialogue. First is the Initial Greeting where the first speaker specified in the Dialogue (*DialogueSpeaker*), is going to greet the *DialogueHearer*. This is done by saying "Hello" to each other. A sample greeting is "Hello Paul", said Mary. After this, the hearer will greet the speaker back. The proper sentence specifications are formed with the exchange of greetings with each other.

Closing utterances have a similar format to that of the Greeting. However, there are two possible closing types: Goodbyes and Thank you. Goodbyes are used when the main character moves to a different location. The main character says goodbye to those that he/she has met in his/her previous location. The Thank you closing is used in relation with the *NeedObject* requirement. Once the main character has gained the object he/she needed through asking another character, they properly say "Thank you" and "You are welcome" to each other.

4.7 Sentence Specifications and Surface Realizer

Throughout creating the story, the Story Planner will create abstract sentence specifications that define the sentences in the story. The medium that is used by the Story Planner in creating the Sentence Specifications is using the Utterance units discussed earlier. The Utterance units are the ones who contains the necessary programming that allows them to create and modify the appropriate Sentence Specifications given the information that has been given to them by the Story Planner / Dialogue Handler. Using this sentence specifications, the Surface Realizer will then interpret each sentence specification and convert them into

Table 4.21: Sentence Specifications Elements

Element Name	Description	Format
Agens	Specifies the Subject / Doer of the action in the sentence	<Specifiers1><Agens Name1>#<Specifiers1><Agens Name2>#<Specifiers1><Agens Name n>;<Agens Modifier1>#<Agens Modifier2>#<Agens Modifier n>
Action	Specifies the Action in the sentence	<ActionName> ; <isNegated> ; <ActionModifier>
Patiens	Specifies the Predicate / Receiver of the action in the sentence	<Specifiers1> <Patiens Name1>#<Specifiers2> <Patiens Name2>#<Specifiersn> <PatiensName n>;<Patiens Modifier 1>#<Patiens Modifier2>#<Patiens Modifier n>
Tense	Specifies the tense of the sentence	Possible values: past, present, future
Interrogative	Specifies if the sentence is a question as well as the type of question	Possible values: how, howmany, what, where, whoindirect, who, whosubject, why, yesno
Complement	Anything that comes after the verb and object of the phrase	<Specifiers> <Complement name>
Location	Specifies the location the action takes place	<Location Name>
LocationModifier	Specifies a modifier to the location	<Location Modifier>

actual English sentences. **Tables 4.21** and **4.22**are the elements that form a Sentence Specification:

The *Agens* specifies the Subject / Doer of the action in the sentence and has the format <Agens Name1>#<Agens Name2>#<AgensName n>;<Agens Modifier1>#<Agens Modifier2>#<Agens Modifier n>. As in the format, there can be multiple agenses separated by the # character. Additionally, Agens Modifiers can also be specified but must match the number of the Agens stated. The modifiers are matched depending on the agens that they match in number. Agens also has an additional specification, the <Specifiers>. Examples of specifiers are 'the', and 'a'.

The *Action* specifies the Action in the sentence and has the format <ActionName> ; <isNegated> ; <ActionModifier>. Similar to the Agens element, it can also have an Action Modifier appeneded to it. Also, another speci-

Table 4.22: Sentence Specifications Elements

Element Name	Description	Format
Dialogue	Specifies if the sentence is a spoken statement by a character	boolean
DialogueType	Specifies the dialogue type used. Affects the tag line of the dialogue	Possible values: can, statement, inforequest, answer, greet, close, agreement, disagreement, added, repeatedsuggestion, closeresponse
Speaker	Specifies the name of the speaker of the utterance	<Speaker name>
Hearer	Specifies the name of the hearer of the utterance	<Hearer name>
AnswerType	Special parameter set if the utterance is a "IS;Why" dialogue	effectOf
Answer	Special parameter set if the utterance is a "IS;Why" dialogue	<cause name>
Negation	Specifies if the statement is negated	true
Instrument	Specifies the instrument used to perform the action	<Instrument name>
VerbForm	Specifies the form of the action	Possible values: BareInfinitive, Gerund, Imperative, Infinitive, Normal, PastParticiple, PresentParticiple
Subject Plural	Specifies if the subject is pluralized	Possible values: true
Object Plural	Specifies if the object is pluralized	Possible values: true
LocationFirst	Specifies if location comes first before the other complements	

Table 4.23: Specifiers

Special Character	Description
@	Sets a 'to' specifier
*	Sets a 'with' at the end of the patiens
!	Sets a 'a' or 'an' specifier
#	Sets a 'part of' specifier
$	Sets a 'made of' specifier
%	Sets a 'the' specifier
˜	Sets a 'with' specifier
?	Sets a 'about' specifier
+	Turns pronomilisation off

fication is appended to it, the <isNegated> which can have the value of "true" or a blank space. Having the value of 'true' indicates that the action to be done is negated. An example of this is the action play. Instead of "Paul played", it will become "Paul did not play".

The *Patiens* specifies the Predicate / Receiver of the action in the sentence and has the format <Specifier> <Patiens Name1> # <Specifier> <Patiens Name2> # <Specifier> <PatiensName n> ; <Patiens Modifier 1> # <Patiens Modifier 2> # <Patiens Modifier n>. Similar to the Agens element, it can also contain multiple patiens and patiens modifiers. Patiens also has an additional specification, the <Specifiers>. Examples of specifiers are 'the', and 'a'. A specifier is represented by a special character. **Table 4.23** is the complete list of the possible specifiers:

Each specifier is used in differing context. The 'to' specifier is usually used together if the patiens concerns a location. An example is "to park". The * specifier is used when the satisy minimum characters fail. An example is "Paul does not have somebody to play *with.*". The 'a' and 'an' specifier can be used for Inquiry dialogues with 'isA' relation. An example is "Police Officer is *a* Job" for words starting with consonants and "Wolf is an animal" for words starting with vowels. The 'part of' and 'made of' specifiers are similar to the 'a' specifier but in relation to the 'part of' relation and 'made of' relation. The 'the' specifier is used in relation if the patiens is a concept retrieved from the *capableOf* relation. An example is "Paul brushed *the* windows". The 'with' specifier can be used for two purposes: (1) when specifying that a concept is used to do the action, ie. an instrument (ex. Paul brushed the windows *with* water). (2) Is used in an invitation of a character to have another character do something with him / her (i.e "Can you come play *with* me?"). Finally, the 'about' specifier is used in Inquiry dialogues. An example usage of 'about' is "We will learn *about* animals". The 'about' specifier is only used in CharHear and CharHear2 Inquiry dialogues because it produces wrong grammar when used with CharAction Inquiry dialogues (ex. We see *about* animals). Finally, the '+' special character turns off the pronomilisation in the Complements processing. This is to avoid creating sentences such as "I will come with you *You*" as opposed to the correct sentence "I will come with you *Erika*".

The *DialogueType* element specifies the Dialogue type to which the Sentence Specification was created in. It also helps the Surface Realizer create the appropriate tag line depending on the Dialogue Type. The possible values of the Dialogue type element are: can, statement, inforequest, answer, greet, close, agreement, disagreement. The "Can" and "Inforequest" dialogue types adds a tag line of " asked " to the utterance. The "Statement" and "Close" dialogue types adds a " said" tag line to the utterance. The "Answer" utterance adds a " answered" tag line to the utterance. The "Greet" dialogue type adds a " greeted" tag line. The "Agreement" dialogue type adds a " agreed" tag line. And finally, the "Disagreement" dialogue type adds a " disagreed" tag line.

The *AnswerType* is a special kind of element that is only used for one function, and that is to identify if an "IS;Why" utterance has happened and is requesting an answer. The AnswerType is set in the *Answer* utterance in an "IS;Why" dialogue. Further discussion will be made of how this is used in the Surface Realizer section.

The *Answer* is also a special kind of element that is used in conjunction with the *AnswerType*. It contains the answer to the "IS;Why" standing question and is used by the Surface Realizer to generate the correct sentence. This will also be discussed later in the Surface Realizer section.

Surface Realizer Algorithm

Below is a pseudocode that defines the steps taken by the Surface Realizer in creating english sentences from the given Sentence Specifications:

```
surfaceRealizer(SentenceSpecifications)
{
    for i = 0, i < SentenceSpecifications, i++
    {
        Get all the elements of current
        SentenceSpecification being processed

        Set Agens, Action, Patiens
        Set Tense
        Add complements
```

```
            Specify Location

            if Negated is true
                 Negate sentence

            Perform final adjustments
      }
}
```

The first thing that the Surface Realizer does it to get all the Sentence Specifications created by the Story Planner and iterate through each Sentence Specifications individually to perform processing.

The first thing it does to a Sentence Specification is to traverse all the Specified Elements in the Sentence Specification and store it for later use.

After it has acquired all the elements, the first thing it does is to create a new instance of *SPhraseSpec*. This is the main phrase element to which all the processing will be done to.

The first processing that the Surface Realizer does to the SPhraseSpec is to set the *Subject, Action, and Object* with the Agens, Action, and Patiens respectively. The first to be set is the the Subject of the phrase using the Agens. Given the Agens element with the format <Agens Modifier1>#<Agens Modifier2>#<Agens Modifier n>, the Surface Realizer first separates the Agens-Modifier from the Agens names. Depending on the number of agenses, it then creates either a *NPPhraseSpec* for a single agens, or uses a *CoordinatedPhraseElement* if there are multiple agenses. While doing this, it also assigns the necessary modifiers to each agens.

Additionally, the Surface Realizer also performs two kinds of pronomolisation: (1) During dialogues and (2) In simple sentences.

For the pronomolisation during dialogues, there are three (3) cases that are checked: (1) If the agens is <All>, then the Surface Realizer sets the *Pronomolisation* feature to a pronoun with first person set (ex. "We"). If the agens is equal to the name of the speaker, the Surface Realizer sets the Pronomilisation to a pronoun of the *First Person* form (ex. 'I'). If the agens is equal to the name of the hearer, then the form of the pronoun is set the *Second Person* form (ex. 'You').

For the pronomolisation in Simple Sentences, the Surface Realizer checks if the Agens has already been stated explicitly on the previous Sentence Specification.

Since the Agens is usually the name of a character in the story, the Surface Realizer checks the gender of the character and performs the proper pronomilisation in the *Third Person* perspective (ex. 'He', 'She'). Additionally, once a dialogue starts, the Agens needs to be explicitly stated again in order for pronomilisation to occur again.

After the Subject has been set by the Surface Realizer, the *Verb* will then be set using the *Action* element. The action element has the form <ActionName> ; <isNegated> ; <ActionModifier>. Also, it uses the *VerbForm* element in order to set the Form of the Verb. In this process, the Surface Realizer sets the Verb using a *VPPhraseSpec*. It then adds to the VPPhraseSpec the ActionModifier, sets the Feature for the VerbForm and if the Verb is Negated, sets the NEGATED flag to true which will be used later. It is worth mentioning here that additional processing is done to the Action Modifier. An 'ly' is added to the name of the Action Modifier. The reason for this is that most adverbs (modifiers to adjectives) have the subscript 'ly' to define the action they are modifying. An example is given the modifier 'quick'. If it is used solely, an incorrect sentence is produced such as "Paul *quick* passed the ball" as opposed to "Paul *quickly* passed the ball".

After the Action has been set by the Surface Realizer, the *Object* or *Patiens* will then be set. It has the form <Specifiers1> <Patiens Name1> # <Specifiers2> <Patiens Name2> # <Specifiersn> <PatiensName n>;<Patiens Modifier 1> # <Patiens Modifier2> # <Patiens Modifier n>. The process in setting the Object is identical to that of setting the Subject, but the only difference is that there are Specifiers that needs to be added to the Patienses. This is done by iterating through each Patiens (if there are many), and converting the found special character to the correct specifier name and calling *setSpecifier* on the NPPhraseSpec that defines the Object.

After the Subject, Verb, and Object has been set, the Surface Realizer then sets the tense of the SPhraseSpec using *Feature.TENSE*.

After setting the tense of the SPhraseSpec, the Surface Realizer then sets the Interrogative Type of the SPhraseSpec if there are any specified, using the *Feature.INTERROGATIVE TYPE*.

After setting the Interrogative Type of the SPhraseSpec, it will then add the specified *Complement* elements to the SPhraseSpec using Add Complements. Complements are anything that comes after the verb and object, and is usually used in this case to add supporting words to complete the sentence. There can be multiple Complements specified in a single Sentence Specification and each complement will be added to the SPhraseSpec. In adding the complement, it follows the similar process with that of setting the Patiens, but with differences.

The first differecne is that complements cannot have modifiers as of the moment. Another difference is that another special character is considered when checking the specifiers of the Complement. This is the special character '+' that turns off pronomilisation in Complements. It is useful in avoiding creating wrong sentences such as "I will play with you *You*" as compared to the correct sentence "I will play with you *Erika*". Another difference is that the *addComplement* function is called by the SPhraseSpec.

Another type of complement is the *Instrument* complement. This complement is automatically added with a 'with a' specifier. Also, multiple instruments can be set and the Surface Realizer will automatically append a 'and a' to the succeeding instruments from the first.

After setting the Complements and Instruments, the Surface Realizer will then set the *Location* of the story. The Location is specified using a PrepositionPhrase or *PPPhraseSpec*. Location automatically have a specifier of 'the', except if the background is "Home" (because referring to one's home as "the Home" is not correct). However, two preposition can be set for the Location: a "to" specifier and an "at" specifier, which is the default specifier used. The "to" preposition is specified by the Story Planner as a parameter if the characters in the story changed location. An example is "Paul and Mary walked *to* Living room". The "at" preposition on the other hand is used when defining that the action is performed in a certain location (ex. Paul played *in* the living room). The Surface Realizer then adds the PPPhraseSpec specified location as a complement to the *SPhraseSpec*.

After setting the Location, if the *NEGATED* flag is set, then the Surface Realizer negates the SPhraseSpec by calling *Feature.NEGATED* and setting it to true.

Finally, the Surface Realizer performs the final adjustments needed to the completed SPhraseSpec. There are several adjustments that are notable here: (1) Check if the dialogue triggered an IS;Why question and perform adjustments, (2) Add additional white spaces every time the story transitions from narrative sentences into a dialogue, and (3) If the Sentence Specification is an utterance in a dialogue, then the SPhraseSpec must be presented in quotation marks and added a tag line depending on the type of utterance.

The first additional adjustment made is if the dialogue triggered an IS;Why question. Particularly, the adjustment will be made to the *Answer* to the IS;Why question. The *Answer* to the IS;Why question will have a Sentence Specification with the elements *AnswerType* and *Answer*. The AnswerType defines that it needs to be formatted as an *Answer* to a *IS;Why* question while the *Answer* element contains the needed answer or cause that will be answered to the *IS;Why*

Table 4.24: Tag lines

Type of dialogue specification	Tag line
statement, Close	said <speaker>
added	added <speaker>
repeatedsuggestion	persuaded <speaker>
inforequest, can	asked <speaker>
answer	answered <speaker>
greet	greeted <speaker>
closeresponse	replied <speaker>
agreement	agreed <speaker>
disagreement	disagreed <speaker>

question. The IS;Why question is a special case because since it wants to find out the 'cause' of an action, it inherently needs to combine two (2) sentences into one. An example is given the IS;Why question "Why does you cry?". The answer would need to be "I cry *because I feel sad.*". In order to create this sentence, another *SPhraseSpec* is used in order to properly create the *Answer* to the IS;Why question. A *CoordinatedPhraseSpec* is then used to create the complete sentence with the form <Sentence 1> "because" <Sentence 2>, where Sentence 1 is the action to which the IS;Why question is concerned. An example is "I cry *because I feel sad.*".

The second additional adjustment made is to add additional white spaces whenever the story transitions from narrative sentences to dialogues. This is in accordance to making the story "easier" to read as stated in the dialogue writing strategies of Ramet (2001) and Dils (1998). This is done by setting a flag every time a dialogue occurs and checking the flag if a dialogue is currently occurring.

The last additional adjustment made by the Surface Realizer is in converting the SPhraseSpec to an utterance or a statement with a tag line. This is done by checking the *Dialogue* element if it is set to true. The first thing the Surface Realizer does is to *realize* the SPhraseSpec and store it into string. It then surrounds the string in quotation marks to specify it is a spoken sentence. Using the *Speaker* element and the *DialogueType* element, the proper tag line is then set. There are a total of 9 tag lines used by the system. **Table 4.24** gives the types of dialogue and its associated tag lines.

This completes the requirements needed in order to specify that a sentence is a spoken sentence by a character. If the Sentence Specification is not in a dialogue, then the Surface Realizer just realizes it.

Finally, the realized sentence (whether it is a Narrative Sentence or a Utterance) will be passed on to the User Interface to be printed out to the user to read.

Additionally, a Sentence Specification may be composed of two or more sentences that will be combined in order to be generated as a single sentence. In order to do this, the Sentence Specification contains the format <Sentence Specification1> & <Sentence Specification 2> & <Sentence Specification n>. Each Sentence Specification is then converted into its completed SPhraseSpec. Once all the Sentence Specifications has been converted to a SPhraseSpec, the first Sentence in the Sentence Specification will be created a new SPhraseSpec. The subsequence Sentences will then be combined to it using the *addComplement* function of the first SPhraseSpec. The Surface Realizer, simpleNLG will automatically handle the combination of these sentences.

Finally, the realized sentences are passed on to the User Interface and printed out and shown to the user.

5 Design and Implementation issues

5.0.1 Dialogue Realizer

SimpleNLG (Venour and Reiter, 2008) is the surface realizer used for generating the surface form of the story text. However, simpleNLG SimpleNLG can only produce declarative sentences of the form <Subject> <Verb> <Object>. In order to create an utterance unit, first the declarative sentence will be created. An example sentence would be: Paul played basketball. The next step would be to encompass the sentence generated in a double quote, which signifies it is an utterance by a character. The next step would be to identify the character who said the utterance. This can be done by adding a tag line such as said Paul. The next step will then be created by combining the declarative sentence with quotation marks with the tag line. Finally, fixing of pronouns will be done to the sentence to make it more appealing. For example, because the subject is Paul and the speaker is Paul, then the story planner assumes that it can change it to the pronoun I to make the sentence I played basketball,said Paul.

Additionally, depending on which dialogue the utterance was used, the declarative can become a question such as What did Paul play?. A different tag line would also be selected depending on the utterance; in this case, the tag line selected is asked Paul.

In creating questions, simpleNLG already provided a nice API that can create simple questions like What, Where, How, and Why. However, there is no API for the Can question, which is used for request utterances. An example would be Can you give me food? or Can I buy food?. As such, the solution to this was to manually add the Can word at the front of the sentence created by simpleNLG and add a question mark (?) at the end of the sentence, replacing the period.

5.0.2 Presence of Lesson in the Stories

Another design issue is where to include the lesson part of the system. The proponent found out that the best place to place the lesson part is in the Inquiry dialogue, Deliberation dialogue, and the Negotiation dialogue. The reason these dialogues were selected was because these dialogues provided the setting where more information and additional statements could be made in the story while not making the story appear too unnatural.

5.0.3 Types of dialogue adapted

In designing the dialogues, there were some issues that were faced. One particular issue is with regards to the types of dialogue. The issues that were faced include designing how each type of dialogue can convey their purpose.

For the Information-Seeking dialogue, one of the issues faced was how to know what kind of information the character was seeking. In order to solve this, the proponent designed it to specify the wh-type of question it is. It can be a What question, Why question, Where question, Can question, or a How question. What questions usually pertain to information about an object such as isA, part of, made of, and canDo. Why question are questions pertaining to the cause of an action (Ex. Why are you crying?). Where question are questions with the intent to know about the location of an object. Can question are usually questions that are request for something. It can be a request for an object, a permission to do something, or an order. And finally, how questions are questions on how to do a specific action. How questions are usually used with deliberation dialogues when deliberating how to do a particular action. For the Inquiry dialogue, the only issue faced was how to know what the characters wanted to know more about the topic being discussed. The solution created for this was similar to that of the What question, in which the needed information about the object is specified (isA, partOf, madeOf, canDo).

For the Inquiry Dialogue, the only problem faced was that there were some instances that there was no topic selected. In order to solve this, the Story Planner was designed to check if there is a topic first for the Inquiry dialogue. If there is no topic, then the Story Planner will first find an action that can create a topic.

For the Deliberation dialogue, the issue faced was with regards to the concept selected. Since Deliberation dialogue was concerned with how to do an action, the related concept that can be applied can be either a ”Noun” or a Verb. For example, consider the action play. When deliberating how to play, the concepts related to it can be either basketball which is a noun, or ride which is a verb. Since basketball is already a noun, it can be already made a sentence (We played basketball). However, if the concept selected is a verb, then the structure of the sentence will be changed, replacing play with the action selected (We ride). Also, since the sentence is incomplete a patiens or object (noun) that relates to the verb selected will be chosen (i.e We ride a bicycle).

For the Persuasion dialogue, the only issue faced was how to use it. The Persuasion dialogue was chosen to represent an order to do something. This can then be used particularly to suggest a solution or an idea from character to a character in order to create the illusion that the idea came from something, specifically, another character's "knowledge".

Finally, for the Negotiation dialogue, the issue faced was with regards to how to select the possible action / event that the negotiator would propose to the character wanting something. Not all events / actions can be used for negotiations as this might make the story less believable. The solution for this was to specify if an action can be considered as a proposal for a negotiation dialogue. In this way, it makes the stories more believable.

Currently, dialogues are mapped to certain actions / events. One of the design issues identified was the need for a first speaker. The first speaker is needed because depending on who the first speaker is, the purpose of the dialogue may change. For example, if the main character is seeking knowledge of something and he/she wants to ask another character, then the first speaker should be the main character since he / she is the one who is seeking information and should start the dialogue.

Finally, the Eristic dilaogue has been left out because it has been observed that it currently has limited use given the domain of the stories. The Eristic dialogue was speculated to more useful for affecting character disposition towards one another, which falls under the field of Affective text generation, similar to that of the inclusion of Personalities of characters within the stories. Because of the inclusion of character disposition towards one another, the domain of the story may then change (ex. Drama, Romance, etc.).

5.0.4 Utterances adapted from DAMSL

For the purpose of this research, the utterances were simplified to only those that were deemed necessary in order to represent the types of dialogues. This includes the Statement utterance, the Info-request utterance, the Answer utterance, the Agreement utterance, and the conventionals such as the Greetings and Closings.

Some Utterance features were not included because it was deemed that it may introduce many other types of elements that may make the story generation process complicated. An example is the "Communicative Status" where in it specifies if the hearer 'understood' what the speaker has said. Introducing this may need to implement a more complex element that can be called a Dialogue History. The Dialogue History might be useful in order to allow the Story Planner to traverse the utterances spoken and do the appropriate action. Another example is in the agreement, only the "agree" and "reject" has been implemented. Implementing the "agree-part", "reject-part", "maybe", and "hold" utterances may make the scope of the research too large. Finally, the Utterance Features may be more useful if there is a Dialogue History included so as the Story Planner may be able to use the Utterance features more and create / modify utterances using the Utterance features.

Another example is the Influencing-Addressee-Future-Action as well as the Committing-speaker-future-action. The story planner kept the idea of the purpose of the two utterances, but represented them using a combination of the info-request utterance, statement utterance, and agreement utterance. The reason for this is to find the right kind of combination of simple utterances to use depending on the type of dialogue (currently used for Information-Seeking dialogue and Negotiation dialogue).

During design and implementation, one design issue faced was regarding the design of the utterances based on the DAMSL annotation scheme. Some parts of the DAMSL annotation scheme proved to be too large a scope to be included in the system, particularly the agree-part, reject-part, maybe, and hold utterances. The understanding backward-looking function has been merged with the agreement backward-looking function because for the current story generator, it seems that they only perform the same thing. Finally, the Information-level was also not used because the proponent found out that the dialogues can work even without the information-level because the type of dialogue inherently already keeps track of its purpose and the needed elements in the dialogue such as the current topic being discussed.

5.0.5 Similarity of Action Names

Because of the way Actions / Events are stored in the Knowledge Base, they may have similar action names which can lead to confusion / error when the Story Planner tries to access it. In order to prevent this, when referring to Action / Events, four (4) columns are considered: the ActionName, the Location, the DialogueTriggered, and the Patiens. The combination of these four elements define the true purpose of an action / event and is unique among them.

5.0.6 Relations adapted and Changed from ConceptNet

In order to make the selection of properties of concepts more dynamic, the proponent made off with the binary relationships and created another way of how to specify the relation propertyOf. Instead of the propertyOf binary relationship, concepts in the knowledge base all have data corresponding to general properties such as isAppearanceLiving, isAppearanceObject, isSize, isAdverb, isQualityLiving, isQualityObject, and isQualityFood. If the concept can be applied a certain general property, the general property flag is set to 1. There is then a separate table for the properties themselves. They then have data corresponding if they are a property of isAppearanceLiving, isAppearanceObject, isSize, isAdverb, isQualityLiving, isQualityObject, and isQualityFood. Properties also have another data defining if that property is considered a good adjective or a bad adjective. Good adjectives are defined as improving the quality of a concept (ex. delicious) while bad adjectives degrade the quality of a concept (ex. rotten). In this way, storing adjectives is easier and selecting adjectives is more dynamic.

Additionally, certain binary relations were added to the Knowledge Base that were not found in the original concept net. This is to cater to the domain of the story generator and help it be able to create its stories. This includes appliedTo, howTo, and performedWith.

5.0.7 Lexical type of a concept

Because of the inclusion of the lessons, sometimes it is needed that a concept be either a NOUN or a VERB. The first idea in order to solve this was to specify in the knowledge base each concept having a type of word; either a noun, a verb, or an adjective. Doing this would be a very time consuming task given the size of the Knowledge Base. For every new concept entered, a type of word would also be needed to be specified for that concept.

The second idea was to use simpleNLG, through the use of the lexicon of the surface realizer, simpleNLG, which keeps track of the type of word of a word in its lexicon. One issue faced with this is that not all the concepts specified in the Knowledge Base can be found in the lexicon. However, based on observation, most of the concepts not found in the lexicon can be considered as nouns, as they appear to be proper nouns. Therefore, if a certain concept is not found in the lexicon, then the system will automatically consider it as a noun.

Finally, the last idea and the implemented idea was to use the Knowledge Base relations to implicitly define the concept of a word. For example, the relation ”isA” defines both nouns (”Dog” isA ”Animal”). However, the appliedTo relation for example, defines a ”VERB” being applied to a ”NOUN”. An example is ”Brush” is appliedTo ”Windows”.

6 Results and Analysis

This section explains how the system was evaluated. It also includes an analysis of the evaluation.

6.1 Target Evaluators

The target evaluators of this research are Story Writers and Linguists. These evaluators were selected because they are the experts that will be able to evaluate this kind of research. There are a total of 3 evaluators that evaluated the system: Ms. Barbra Pe, a full-time professor in the literature department in De La Salle University, Ms. Jewel Castro, a part-time professor and a script writer of the literature department, and lastly, Ms. Neslie Tan a part-time professor in the English and Applied linguistics department.

6.2 Evaluation methodology

9 stories subject for evaluation were presented to the evaluators, 3 for each character problem (Knowledge, Hunger, Feel). Each story for each problem will have a different lesson specified, one for each possible lesson (Noun, Verb, Adjective). There will be a total of 9 stories with 3 stories for Knowledge (with Noun, Verb, Adjective Lesson), 3 stories for Hunger (with Noun, Verb, Adjective), and 3 stories for Feel (also with Noun, Verb, Adjective). The background for each story can be any Primary background, however, for the 9 stories, the five (5) Primary Backgrounds has to be selected as input at least once. The User Inputs used to form the 9 stories presented are given in Table 6.1. The full story text can be seen in Appendix B.

Table 6.1: Stories evaluated

Story Number	User Inputs used
1	School,Hunger,Noun
2	Park, Hunger, Verb
3	Mall, Hunger, Adjective
4	Home, Knowledge, Noun
5	Zoo, Knowledge,Verb
6	Zoo, Knowledge, Adjective
7	Home, Feel, Noun
8	Park, Feel, Verb
9	Home, Feel, Adjective

6.3 Quantitative Results and Discussion

The quantitative evaluation of the linguists is separated into 3 criteria: **Overall story**, **Dialogue**, and **Grammar**. Overall story is concerned with the overall quality of the story, Dialogue is more concerned with the quality of the dialogues, and Grammar is for the linguistic / syntactic aspect of the sentences in the story. Each general category then have sub-criterias, which are the ones to be rated. **Table 6.2**, **6.3**, and **6.4** shows each criteria with its sub-criterias.

The evaluators rated each criterion for each category on a scale of 1 to 4, with 4 being the highest. 4 means that the criteria is completely present, 3 means that the criteria is present but incomplete, 2 means that the criteria is partially present, while 1 means that it is not present.

Tables 6.5, **6.6**, and **6.7** shows the summarized Quantitative evaluation from the evaluators. The Quantitative Results can be used to determine the areas of the system that are considered poor and therefore, may be improved in future works.

6.3.1 Overall Story Quantitative Analysis

The **Overall Story** received an overall score of 2.72 out of 4. The stories created were acceptable stories but were lacking / performing poorly in some areas that stories have. The stories were also considered basic enough for the target audience and also able to deliver the lessons specified by the user in the story.

In order to give more detail on what areas the stories performed poorly in, the Quantitative evaluation of each criteria will be given.

Table 6.2: Overall Story

Criteria	Description
Plot completeness	Story has introduction, rising action, climax, solution, and conclusion
Characters, Objects, and backgrounds	How well the interplay of characters, objects, and backgrounds is If characters, objects, and setting was properly introduced
Story is coherent	If the story produced was coherent or having a story The actions characters took made sense The events that happened in the story made sense The transitions from event to another event made sense
Story content and User input	The story generated was able to properly convey the user inputs (background, problem, and lesson) The story is appropriate to target age group

Table 6.3: Dialogue

Criteria	Description
Dialogue presence in the story is coherent / believable	Occurrence of dialogues in the story makes sense (by supporting / helps complete the story) Dialogue was easy to understand
Dialogue content	The topic the characters are talking about is coherent between each other and kept into context with the purpose of the dialogue Characters participating in the dialogue are apparent
Dialogue tag lines are used correctly	Tag lines are used to identify which character spoke, and how he/she spoke. Ex. said Paul, asked Paul, etc.
Proper greetings / goodbyes were used correctly	Proper use of greetings / closings

Table 6.4: Grammar

Criteria	Description
Sentences are grammatically correct	For both narrative sentences and dialogue sentences
Word contexts, pronouns, articles are used correctly	Generally concerns sentence structures

Table 6.5: Overall Story evaluation

Evaluator Number	Plot Completeness	Characters, Objects, and Background	Story is Coherent	Story Content and User Input	Overall Total
1	4.00	3.78	4.00	4.00	3.94
2	3.78	2.78	2.22	3.00	2.94
3	1.44	1.22	1.33	1.11	1.27
Total	3.07	2.59	2.52	2.70	2.72

Table 6.6: Dialogue evaluation

Evaluator Number	Dialogue Presence in the story was coherent	Dialogue content was coherent	Dialogue tag lines are used correctly	Proper greetings and goodbyes	Overall Total
1	3.44	3.56	4.00	4.00	3.31
2	2.78	2.55	3.56	3.89	3.19
3	1.56	1.56	2.22	2.56	1.91
Total	2.59	2.56	3.26	3.48	2.80

Table 6.7: Grammar evaluation

Evaluator Number	Sentences are grammatically correct	Word context, Pronouns, Articles are used correctly	Overall Total
1	3.00	3.00	3.00
2	3.00	3.00	3.00
3	1.67	1.44	1.56
Total	2.55	2.48	2.52

The **Plot Completeness** was rated 3.07 out of 4. The stories were able to have the plots that make up a story. However, there were still some lacking elements in the story.

One of the reason that brought down the grade of the Plot Completeness was that there was no obvious climax in the stories as one of the evaluators noted. Because of that, it hinders the plot completeness aspect of the stories created by the system. The reason for the stories having no obvious climax was because of how the events are stored in the Knowledge Base and how the Story Planner works. The events in the Knowledge Base contain information if it can create the three (3) character problems, specifically, the CreateHunger, CreateKnowledge, and CreateFeel. It also contains information if it can SolveHunger, SolveKnowledge, and SolveFeel. These information are used for guiding the Story Planner in selecting the Inciting Incident, and the Falling Action / Resolution respectively. However, there is no information that specifies the Climax. The nearest thing that the Story Planner creates that is similar to the climax is the *requirements* needed in order to perform the event, particularly: (1) Going to the location of the event (2) Executing first events of the event if any (3) Satisfying the minimum character requirements of the event and (4) Satisfying the Requirements of the event (askDo, NeedObject). For future work, more information of how to model the Climax may be researched into.

The **Characters, Objects, and Backgrounds** criteria received a score of 2.59 out of 4. The reason for this is that some events in the story are quite confusing / can be improved with regards to the Characters, Objects, and Backgrounds.

One of the reasons that affected brought down the score of the Characters, Objects, and Backgrounds was that better adjectives and verbs should have be chosen when discussing about verbs or objects. For example, one of the sentences in the story give the sentence "*He thoroughly made small pancake.*" seen in *Story 8 (Home, Feel, Adjective)*. The adjective thouroughly may be replaced with a better adjective, such as "carefully". Another example is the sentence "Ballpen is nice" in *Story 3 (Mall, Hunger, Adjective)*. The adjective "nice" may be replaced with an adjective that may describe the object ballpen more (ex. sharp, pointy, etc.). The reason that these story sentences were created was because of the way the Modifiers / Adjectives are stored in the Knowledge Base. Properties are stored in the Knowledge Base as being applicable adjectives to appearance of Living Things, appearance of Objects, size, can be an adverb, abstract quality of living things, abstract quality of objects, and abstract quality of foods. Each object then also has a map to the applicable adjectives that it can have. For example, the object "Ballpen" can have an adjective of abstract quality of an object, and the adjective "nice" can be an adjective for an abstract quality of an object, then it can be selected by the Story Planner. This may say that although

Table 6.8: Consider better verbs

Story snippet	Remarks
[1] Michelle, Farm keeper Joe and Zoo Keeper Roger saw animal.	[1] Michelle, Farm keeper Joe and Zoo Keeper Roger saw animal.
[2] "I see a lion.", answered Zoo Keeper Roger	[2] "I see a lion.", answered Zoo Keeper Roger
[3] "Lion cans roar.", said Michelle	[3] "Lion cans roar.", said Michelle
[4] "I see a wolf.", answered Zoo Keeper Roger	[4] "I see a wolf.", answered Zoo Keeper Roger
[5] "Wolf cans run.", said Michelle	*[5] "Wolf can howl.", said Michelle*
[6] "I see a giraffe.", answered Zoo Keeper Roger	[6] "I see a giraffe.", answered Zoo Keeper Roger
[7] "Giraffe cans run.", said Michelle	[7] "Giraffe cans run.", said Michelle

dynamic specification of modifiers to concepts may provide greater variety, but less accurate and satisfying results compared to manually identifying the adjectives of concepts.

Similar to the adjective issue, a similar issue raised was with regards to the verbs used in the Story. The evaluators suggested that the verbs also provide greater variety but with Uniqueness. **Table 6.8** gives an example in the story snippet of *Story 5 (Zoo, Knowledge, Verb).*

The verbs associated with the animals Wolf and Giraffe might be improved to provide greater lexical variety as well as to give more knowledge on the distinguishing feature of the animal. For example, instead of "Wolf can run", it can be "Wolf can howl". Since the relation used here (canDo) is only a binary relation, this has already been addressed in the current system.

Another reason that also brought down the rating of the Characters, Objects, and Backgrounds was that the Characters go to weird locations and therefore, some locations are confusing. An instance of this is given the Story Snippet below taken from *Story 2 (Park, Hunger, Verb)*:

```
[1] Michelle and Paul will skip jump rope.
[2] Michelle became hungry.

[3] "Goodbye Paul.",  said Michelle
[4] "Goodbye Michelle.",  said Paul

[5] Michelle walked to Home.
[6] Michelle walked to Bathroom.
[7] She meets Mommy Emma and Daddy Will in Bathroom.
[8] Mommy Emma talked to Michelle.
```

```
[9] "Hello Michelle.",  greeted Mommy Emma
[10] "Hello Mommy Emma.",  greeted Michelle
[11] "I am hungry.", said Michelle
```

Since Michelle is hungry, her first trip should be in the Kitchen where there may be food. However, in this story, *line 6* shows that Michelle walked to the Bathroom to talk with Mommy Emma.

The reason for this is because of the way the Story Planner works. The event selected by the Story Planner was the ”Suggest Solution” event because it has a SolveHunger specification. The ”Suggest Solution” event has another character tell the main character a solution to his / her problem, in this case, Hunger. In order to do this, the main character needs to find another character to tell him / her. Since characters are randomly assigned to a location, it just so happens that Mommy Emma is in the Bathroom.

Another reason that affected the rating was that the Primary Background that was specified as an input was sometimes confusing because of the story. For example, in *Story 4 (Home, Knowledge, Noun)*, even though the specified background was Home, the characters went to the mall, and it seems that they spend most of their time there. The reason for this is because of the way the Story Planner and the Knowledge Base works. The Knowledge Base has an event in home called ”go shopping” which creates the Knowledge problem, which is the Character problem specified in the story above. Since the ”go shopping” event happens in Home, the Story Planner already considers that the ”Home” primary background was satisfied. Additionally, the ”go shopping” event triggers a special perform *Prox(Mall)*, which requires that the characters go to the location ”Mall” at the end of the event.

The **Story Coherency** criteria scored the lowest out of the overall story, with a score of 2.52 over 4. Below is a discussion of the reason why this criteria received that score.

One of the reasons that brought down the rating of the Story Coherency is that the goal in ”Knowledge” problem stories were unclear. For example, given *Story 4 (Home, Knowledge, Noun)*, one of the evaluators noted that it is unclear why the characters went to the mall to see appliances. An improvement is to include an event where the characters has *reason* to go to the mall (ex. buying new appliances because the old appliances broke down). Again, this also points that a ”theme” may be more useful instead of character problems when doing this.

The *Story 5 (Zoo, Knowledge, Verb)* is considered as an improvement compared to the Story 4 (Home, Knowledge, Noun). It is considered an improvement by the evaluators because the story has the goal of taking a trip to the zoo and see the animals, which can already set a plot for the story. This is different in the previous story where the event "go shopping" does not appear to be realistic by only itself.

Another reason that affected the coherency of the stories was that some actions appear confusing as to the motive of the character. An instance of this is given the Story Snippet below taken from *Story 2 (Park, Hunger, Verb)*:

```
[1] Michelle and Paul will skip jump rope.
[2] Michelle became hungry.

[3] "Goodbye Paul.",  said Michelle
[4] "Goodbye Michelle.",  said Paul

[5] Michelle walked to Home.
[6] Michelle walked to Bathroom.
[7] She meets Mommy Emma and Daddy Will in Bathroom.
[8] Mommy Emma talked to Michelle.

[9] "Hello Michelle.",  greeted Mommy Emma
[10] "Hello Mommy Emma.",  greeted Michelle
[11] "I am hungry.", said Michelle
```

Instead of just going to the Kitchen to get food, Michelle takes the time to talk with Mommy Emma which can be seen in *lines 6 - 10.* The reason for this is because the event selected by the Story Planner to solve the "Hunger" problem was the "Suggest Solution" event where another character is going to tell the main character a solution to his / her problem. Therefore, Michelle takes the time to go to another character that can suggest a solution to him / her.

Another factor that brought down of the Story Coherence is because the causal relationships between consecutive events are not clearly established. In some stories, one event is followed by another event, which seems completely unrelated. This causes confusion in the read, who will not grasp the storys meaning / message. The possible reason for this is because of the weak connection of actions / events in the knowledge base to each other. They are only explicitly connected by the hasFirstSubEvent and the NextEvent specifications. Aside from this, there are no more explicit connections between events. Another connection that may be considered is the connection of events using the problem they create or solve.

For example, actions that CreateHunger are followed by those that SolvesHunger. The same goes for Knowledge, and Feel. However, this connections are not direct. They are only connected because of the algorithm of the Story Planner in creating the story, in which it first selects an event that CreateProblem (Inciting Incident) and after it has found and executed the necessary events, it selects another event that can SolveProblem. Because of this, there are no explicit transitions that link the events from the 1st half (Inciting Incident) to the 2nd half (Falling Action). These connections are not direct and therefore can be considered to be weak connections. An example of this is in *Story 1 (School, Hunger, Noun)* which is defined below with markings of the 1st half (Exposition and Inciting Incident) and the 2nd half (Climax, Falling Action, Conclusion):

```
The day is rainy.
Erika is in School.
She is in Classroom.
She has class in Classroom.
She meets Michelle and Teacher Lady in Classroom.
She talked to to Michelle and Teacher Lady.

"Hello Michelle and Teacher Lady.",  greeted Erika
"Hello Erika.",  greeted Michelle
"Hello Erika.",  greeted Teacher Lady
"What does we do?", asked Erika
"We will learn.", said Teacher Lady
"What does we learn?", asked Erika
"We learn about food.", answered Teacher Lady

Erika, Michelle and Teacher Lady learned food.

"Spaghetti is food.", answered Teacher Lady
"Fruit is food.", answered Michelle
"Pizza is food.", answered Teacher Lady

Erika became hungry.

----------END OF 1ST HALF-----------
----------START OF 2ND HALF----------

"Goodbye Michelle.",  said Erika
"Goodbye Erika.",  said Michelle

Erika walked to Cafeteria.
Erika wants to eat.
She needs food to eat.
She meets Dennis, Paul and Cafeteria Lady Elise in Cafeteria.
She talked to Cafeteria Lady Elise.

"Hello Cafeteria Lady Elise.",  greeted Erika
"Hello Erika.",  greeted Cafeteria Lady Elise
"Can I buy food?", asked Erika
"You will buy food Erika.", Cafeteria Lady Elise
"What does you sell?", asked Erika
"I sell a milk.", answered Cafeteria Lady Elise
"I sell a egg.", answered Cafeteria Lady Elise
"I sell a hamburger.", answered Cafeteria Lady Elise
"I sell a pizza.", answered Cafeteria Lady Elise

Erika bought hamburger.

"Thank you Cafeteria Lady Elise.",  said Erika
"You are welcome Erika.",  said Cafeteria Lady Elise

Erika eats hamburger in Cafeteria.
Erika felt tired.
She went home.
She learned many things that day.
She learned different food in Classroom.
She learned different food in Cafeteria.
```

In the story above, it can be seen that there is a weak connection between the 1st half and the 2nd half. Although it is implied that Erika went to the Cafeteria to get some food, it is not very clear and the transition from 1st half to 2nd half is not very good.

Another example of problem with transitions is given the story snippet below from *Story 7 (Home, Feel, Noun)*:

```
[1] "What does we do?", asked Michelle
[2] "We will play.", said Dennis
[3] "What does we play?", asked Michelle
[4] "We will play marble.", answered Dennis
[5] "We will play toys.", answered Michelle
[6] "We will play robots.", answered Dennis
```

```
[7] "We will play hide and seek.", answered Dennis

[8] Michelle chose to play hide and seek.

[9] "We will play hide and seek.", said Michelle
[10] "We will play hide and seek Michelle.", Dennis

[11] Michelle and Dennis played hide and seek.
[12] Michelle gets hurt in Living room.
[13] She felt sad.
[14] She cries in Living room.
```

In this story, there is a lost connection with what caused Michelle to get hurt which can be seen in *line 12.* Although it is implied that she got hurt because of playing, there is no clear statement as to what caused it. In order to solve this, other events may be added as to why Michelle got hurt (ex. Falling down while running).

Finally, the **Story Content and User input** received a score of 2.70 out of 4. The Story generator was still quite able to generate a story that adheres to the input of the user concerning Background, Problem, and Lesson.

However, the rating was affected by sometimes confusing Primary location changes such as the example before in which the Primary Background was "Home" yet the characters went to the "Mall".

Another concern was with regards to the choices of concepts to discuss ("Food" and "Healthy foods"), verbs discussed ("Wolf can run" -> "Wolf can howl"), and the adjectives selected ("Ballpen is nice").

Also, the evaluators noted that some parts in the story were quite a bit confusing. For example, given the story snippet given below from *Story 2 (Park, Hunger, Verb)*:

```
[1] "Can I have food?", asked Michelle
[2] "You will have food Michelle.", Mommy Emma
[3] "What does you give?", asked Michelle
[4] "I give a water.", answered Mommy Emma
[5] "You can drink water.", said Mommy Emma
[6] "I give a vegetable.", answered Mommy Emma
[7] "You can chop vegetable.", said Mommy Emma
[8] "I give a egg.", answered Mommy Emma
```

```
[9] "You can crack egg.", said Mommy Emma
```

The Story Planner is just having the characters discuss about what you can do with water, vegetable, and egg. Although this satisfies the lesson "Verb", the motive of the discussion about why they are discussing about what you can do with the foods is unclear. The reason that the Story Planner does this is because it just provides additional statements that is added to a discussion given the Lesson specified by the user. A future recommendation may be to improve upon this in order for the Story Planner to be able to define a motive for their discussion.

6.3.2 Dialogue Quantitative Analysis

The **Dialogue** received an overall score of 2.80 out of 4. In order to give more detail on what areas the stories performed poorly in, the Quantitative evaluation of each sub-criteria will be given.

The **Dialogue presence in story is coherent** received a score of 2.59 out of 4. Several factors that affected this rating will be discussed below.

One of the factors that brought down the rating of this criteria is the Story to Dialogue transition. Some dialogues appear unnatural / unrealistic because characters start speaking to each other without clear motives. This raises the question of why the characters started talking. For example, given the story snippet below from *Story 4 (Home, Knowledge, Noun)*, in *line 3*, Dennis immediately talks with Cashier Troy with no motive being stated.

```
[1] Dennis and Mommy Debbie went to Department store.
[2] Dennis meets Mommy Debbie, and Cashier Troy in Department store.
[3] He talked to Cashier Troy.

[4] "Hello Cashier Troy.",  greeted Dennis
[5] "Hello Dennis.",  greeted Cashier Troy
[6] "What does we do?", asked Dennis
[7] "We will see.", said Mommy Debbie
[8] "What does we see?", asked Dennis
[9] "We see a appliances.", answered Cashier Troy
```

The reason that this appears in the stories is because dialogues are triggered by the Story Planner if it checks that there is a dialogue mapped to the current action / event to be performed. In the story above, the event "see", triggers an "Information-Seeking" dialogue. However, the Story Planner is currently unable to know the main reason why the dialogue was triggered in the first place (or even why that type of dialogue is mapped to the action / event. The dialogues were manually mapped to the action / events by the proponent depending on his own judgement of what dialogue is to be used for that event.

In order to improve this, a possible future work is to define an algorithm that will help the Story Planner understand more on when to trigger certain types of dialogues. In doing this, it may also be able to provide the necessary transitions in the form of dialogue or simple sentences, that will give more detail as to the purpose of the start of the dialogue.

The **Dialogue content coherency criteria** was given a rating of 2.56 by the evaluators. Below is a discussion of the reason why the evaluators gave this rating. Most of the issues the evaluators raised are concerned with suggestions / improvements of what can be added into the dialogue. However, they did not have a concern with the way the dialogue went, implying that the correct utterances were used in order to form the main types of dialogues.

One of the factors that affected the rating of this criteria was because wrong characters sometimes answered / participated in the dialogue at the wrong time.

For example, given the earlier story snippet above from *Story 4 (Home, Knowledge, Noun)*, Mommy Debbie should have been the one to answer in *line 9*, instead of Cashier Troy. The reason that Cashier Troy was selected was because Static Characters are also considered by the Story Planner as Parent Characters, which the "see" event has a Dialogue Hearer (<PC>) specification.

Another example is given the story snippet below from *Story 7 (Home, Feel, Noun)*. In *line 5* Dennis should be the one to suggest the things they play because Michelle is the one asking about the things they can play:

```
[1] "What does we do?", asked Michelle $<$- Main character
[2] "We will play.", said Dennis
[3] "What does we play?", asked Michelle
[4] "We will play marble.", answered Dennis
[5] "We will play toys.", answered Michelle
[6] "We will play robots.", answered Dennis
[7] "We will play hide and seek.", answered Dennis
```

Finally, another example story snippet from *Story 3 (Mall, Hunger, Adjective)* is given below. Instead of Erika stating additional information about the objects they are discussing in *lines 3, 5, and 7*, a more knowledgeable person (Cashier Edna) may be the one to state it.

```
[1] Erika, Mommy Len, Daddy John, and Cashier Edna saw writing materials.
[2] Erika sees that pencil is writing materials.

[3] "Pencil is handsome.", said Erika
[4] Erika sees that eraser is writing materials.

[5] "Eraser is small.", said Erika
[6] Erika sees that ballpen is writing materials.

[7] "Ballpen is nice.", said Erika
```

This issues has already been addressed in the current state of the system in order to conform with the suggestion of the evaluators. However, the design of who the speakers and hearers in the dialogue right now are hard-coded into the knowledge base by being mapped to the Actions / Events. This design makes the inputting of dialogue participants in actions / events to be quite static and time-consuming as they are manually added. An improvement might be to design an algorithm for both the Knowledge Base and Story Planner that allows the Story Planner to understand who the dialogue participants will be in an event.

For the criteria of the **Dialogue tag lines are used correctly**, the evaluators gave it a rating of 3.26 out of 4. The evaluators noted that the usage of dialogue tag lines were mostly correct. However, one of the evaluators noted that it would be an improvement if there were more tag lines included in the story.

Finally, the **Proper greetings and goodbyes** criteria received a score of 3.48 out of 4. The evaluators noted that proper greetings and goodbyes were used throughout in the story. This suggests that the Greetings and Closings were properly placed in the Story Planner's algorithm. The Greeting was triggered every time the main character talks to another character for the first time. The Closing was triggered every time the main character moves into a different location / background.

6.3.3 Grammar Quantitative Analysis

The **Grammar** criteria scored a score of 2.52 out of 4. Contributing to this score are the criteria of the grammatical quality of sentences, and the correct usage

Table 6.9: Proper punctuations

Incorrect sentence	Correct sentence
I will come with you, said Erika	I will come with you, said Erika.
"Hello Dennis," said Erika	"Hello, Dennis," said Erika

Table 6.10: Proper capitalization

Incorrect sentence	Correct sentence
Erika is in School	Erika is in *school*
"You will come with me to Playground.", said Michelle	"You will come with me to *playground*.", said Michelle

of word context, pronouns, and articles. These will be further discussed in the sub-sections below. Most of these were already addressed in the system.

The first criteria is the **Sentences are grammatically correct**. The evaluators gave this criteria a score of 2.55 out of 4. Some notable remarks that the evaluators had were concerned with proper punctuation, capitalization, verb tenses, and commas.

First, there were some wrong punctuations in the sentences in the story. First, proper punctuation with the dialogues was incorrect, specifically the position of the comma to define an utterance / spoken sentence by a character. **Table 6.9** shows the incorrect sentences and the corrected sentences. This has been addressed in the current system.

Capitalization was also an issue, specifically, with regards to capitalization of location. Locations were capitalized when they should have been not. **Table 6.10** shows the incorrect and correct sentences. This has already been addressed in the current system.

Some verb tenses were also wrong in the stories. **Table 6.11** shows the incorrect sentences and correct sentences. This and other verb tenses were already corrected with accordance to the suggestion of the evaluators.

The next criteria is the **Word context, pronouns, articles are used correctly**. The evaluators gave this criteria a score of 2.48 out of 4. The evaluators noted some errors in the use / or lack of proper prepositions, articles, and plurality

Table 6.11: Proper verb tenses

Incorrect sentence	Correct sentence
What does we do?	"What will we do?"

Table 6.12: Proper prepositions

Incorrect sentence	Correct sentence
”She learned the different foods in classroom”	”She learned about the different foods in classroom”
”Erika is in home”	”Erika is at home”
”I will go shopping to Mall”	”I will go shopping *at the* Mall”

Table 6.13: Proper articles

Incorrect sentence	Correct sentence
”I sell a milk”	”I sell milk”
She is in Playground.	She is at *the* Playground.

of form of words.

First, there were issues with the correct usage / lack of prepositions. **Table 6.12** shows the incorrect sentences and the corrected sentences. This has been addressed and fixed in the current system.

Another issue was with regards to the proper usage of articles in the story. Some notable examples of article errors are listed in **Table 6.13**.

Finally, the evaluators had an issue with regards to the plurality of words. Some words may produce better results when used in dialogue in plural form. **Table 6.14** gives an example story snippet from *Story 5 (Zoo, Knowledge, Verb)*, with converted *lines 2, 4, and 6* pluralized.

Also, it might be more appropriate to pluralize words when being used in an Inquiry:List Of options dialogue. For example, **Table 6.15** gives a story snippet with the *lines 6, 7, 8, and 9* pluralized.

The main reason that there are some errors with regards to the correct usage of articles and prepositions is because the articles and prepositions are added on Sentence Specifications depending on where they have been called in the Story

Table 6.14: Plural Form

Story snippet	Remarks
[1] ”I see a lion.”, answered Zoo Keeper Roger [2] ”Lion cans roar.”, said Michelle [3] ”I see a wolf.”, answered Zoo Keeper Roger [4] ”Wolf cans run.”, said Michelle [5] ”I see a giraffe.”, answered Zoo Keeper Roger [6] ”Giraffe cans run.”, said Michelle	[1] ”I see a lion.”, answered Zoo Keeper Roger *[2] ”Lions can roar.”, said Michelle* [3] ”I see a wolf.”, answered Zoo Keeper Roger *[4] ”Wolves can run.”, said Michelle* [5] ”I see a giraffe.”, answered Zoo Keeper Roger *[6] ”Giraffes can run.”, said Michelle*

Table 6.15: Plural Form

Story snippet	Remarks
[1] ”Hello Cafeteria Lady Elise.”, greeted Erika	[1] ”Hello Cafeteria Lady Elise.”, greeted Erika
[2] ”Hello Erika.”, greeted Cafeteria Lady Elise	[2] ”Hello Erika.”, greeted Cafeteria Lady Elise
[3] ”Can I buy food?”, asked Erika	[3] ”Can I buy food?”, asked Erika
[4] ”You will buy food Erika.”, Cafeteria Lady Elise	[4] ”You will buy food Erika.”, Cafeteria Lady Elise
[5] ”What does you sell?”, asked Erika	[5] ”What does you sell?”, asked Erika
[6] ”I sell a milk.”, answered Cafeteria Lady Elise	[6] ”I sell milks.”, answered Cafeteria Lady Elise
[7] ”I sell a egg.”, answered Cafeteria Lady Elise	[7] ”I sell eggs.”, answered Cafeteria Lady Elise
[8] ”I sell a hamburger.”, answered Cafeteria Lady Elise	[8] ”I sell hamburgers.”, answered Cafeteria Lady Elise
[9] ”I sell a pizza.”, answered Cafeteria Lady Elise	[9] ”I sell pizzas.”, answered Cafeteria Lady Elise

Planner. For example, take the example below which happens in a Deliberation dialogue.

```
[1] "We will play hide and seek," answered Dennis.
[2] "We will play robot," answered Daddy Will.
[3] "We will play marble," answered Dennis.
[4] "We will play toy," answered Dennis.
```

All the concepts above (Hide and seek, robot, marble, and toy) has been selected by finding a ”conceptuallyRelatedTo” relation with the action ”play”. Also, all the sentences have been generated with the same sentence specification. However, 1 of the sentences is correct while the others are not grammatically correct. The sentence ”We will play hide and seek” sounds correct and is therefore left at it is. However, the last 3 sentences is grammatically incorrect. It should be that they have a ”with” specifier to produce the sentence ”We will play with a robot”. If a ”with” specifier is included in the Sentence Specification used to create the above sentences, it might affect the already correct sentence ”We will play hide and seek” to ”We will play with hide and seek”.

A possible solution to this is to use the relation used in the Knowledge Base to specify the correct article / preposition. For example, the relation ”canDo” can be used to define what instrument is used to do an action. An example is ”knife” canDo ”chop”, can then be used to ”Paul chopped vegetable with a knife”. However, this might need further research on what relations define what articles and prepositions. Additionally, other relations might need to be added in order to define other prepositions / articles.

Additionally, this is not the only solution. The actual word may need to have another special relation that defines what specifier it should have. An example is given below:

```
Paul brushed his tooth (Agens, Action, Patiens)
Paul ate a banana (Agens, Action, Patiens)
"Paul drank milk" (Agens, Action, Patiens)
Paul ran the racetrack (Agens, Action, Patiens)
```

In the example given above, the Knowledge Base should know that a "tooth" is a possession of a human so that the specifier is a possessive specifier. Additionally, additional relations may be added in order to know when to use "a" vs "the" specifier and when to not use a specifier (such as in the case of "Paul drank milk" vs "Paul drank a milk".

Deliberation (Cook) "How will we cook?" asked Dennis. "We will pour a sauce," answered Daddy Harry. "We will make a cereal," answered Daddy Harry. "We will peel a fruits," answered Daddy Harry. "We will chop a vegetable," answered Daddy Harry. "We will cook a noodles," answered Daddy Harry.

Deliberation (Play) "How will we play?" asked Dennis. "We will ride a bike," answered Daddy Harry. "We will kick a ball," answered Daddy Harry. "We will ride a bike," answered Daddy Harry.

6.4 Qualitative Results and discussion

The Qualitative Results are written comments or suggestions given by the evaluators that will help in specifically identifying the areas of the system that are considered poor or wrong, and therefore may be improved in future works. However, not all comments / suggestions were implemented in the current system.

6.4.1 Overall Story Qualitative Evaluation

The stories created were acceptable stories but were lacking / performing poorly in some areas that stories have. The stories were also considered basic enough for

Table 6.16: More proper event names / terms

Story snippet	Remarks
[1] Erika walked to Food court. [2] Erika wants to eat. [3] She needs food to eat. [4] She meets Paul and Cashier Nina in Food court. [5] She does not have someone to ask location with in Food court. [6] She went to Bookstore.	[1] Erika walked to Food court. [2] Erika wants to eat. [3] She needs food to eat. [4] She meets Paul and Cashier Nina in Food court. *[5] She does not have anything there that she likes to eat.* [6] She went to Bookstore.

the target audience and also able to deliver the lessons specified by the user in the story.

The evaluators suggested that the addition of *Themes* might help improve the completeness of the plot of the story. For example, instead of having a story just discussing about "Food" as in *Story 1 (School, Hunger, Noun)*, there can be a theme set such as discussing about "Healthy Foods" which promotes knowledge about healthy foods. The addition of themes may also improve the overall story in the way that the plots are focused on this theme (ex. eating junk foods, having a toothache, and eating "Healthy Foods" instead of Junk foods). However, because of this introduction, other algorithms may need to be introduced in order to properly represent the knowledge and algorithm to handle this.

Another suggestion by one of the evaluators is the addition of more types of the problem "Feel" verbs. The reason for this is to provide variety to the story. Currently, there are 2 kinds of "Feel" problems in the story: Physical Feel problem and Emotional Feel problem. For the Physical problem, there are only a limited number of verbs that are associated with it in the Knowledge Base. These are the "get headache" and the "get hurt" events. A solution to this is to introduce more events that are able to create the Physical Feel problem in order to provide variety in the story.

The evaluators also suggested that **Color properties** may be added to the system because Color is a basic property that can be taught to the target audience, which is children. This has been added to the system.

Another suggestion of the evaluators was to improve the terms assigned to events in the story. In order to show this, **Table 6.16** gives an example story snippet from *Story 3 (Mall, Hunger, Adjective)*:

In *line 6*, it could be changed to "She does not have anything there that she likes to eat". However, this may not be possible with the current design of the

Story Planner because right now, the sentence "She does not have someone to ask location with in Food court" was created by defining the Sentence Specifications: Agens (She or Mary), Negated(true), Patiens(someone), ActionName(ask location), Location(Food court). This sentence was created because of the failure to satisfy the minimum character requirement of the event (ask location). All sentences created when the minimum character requirement satisfaction fails follows the format mentioned above. This is to create other sentences such as "She does not have someone to play with in Playground".

The evaluators also provided suggestions on how to improve the transitions in the story. This includes (1) Emphasize what happens next / connection of sentences to each other, (2) Emphasize shift in feelings, (3) Adding explicit connection of events to one another.

However, in the current design of the Story Planner, it may not be completely done. The concept "hide and seek" which the characters chose to play was selected dynamically by the Story Planner along with "toys", "robots", and "marbles". Therefore, if there is an event that is specified that "playing" leads to "falling down" and leads to "get hurt", if another concept was selected (ex. "robots"), it may not be entirely believable that the character fell down while playing robots. Therefore, this may be done in further research.

Additionally, another suggestion by the evaluators to improve the Story transition is the use of Discourse Markers. Discourse markers can be used to define connections of sentences to each other. **Table 6.17** gives an example story snippet from *Story 1 (School, Hunger, Noun)* in *line 10.*

6.4.2 Dialogue Qualitative Evaluation

The overall quality of the Dialogues was considered acceptable by the evaluators but may be improved with regards to believability and authenticity.

First, the evaluators gave a suggestion that dialogues may become more believable / authentic by introducing themes in the story. This goes back to the example of "Food" and changing it to "Healthy Foods". Additionally, because of the inclusion of a theme, it will affect the plots of the story which in turn will affect the flow of the dialogue and the dialogue content. By applying a theme to the story, the dialogue of why the characters are talking about "Healthy Foods" will be more believable.

Another suggestion of one of the evaluators was to include supporting utterance interjections that may also improve authenticity / believability of the dia-

Table 6.17: Improve transition using Discourse markers

Story snippet	Remarks
[1] "Hello Cafeteria Lady Elise.", greeted Erika [2] "Hello Erika.", greeted Cafeteria Lady Elise [3] "Can I buy food?", asked Erika [4] "You will buy food Erika.", Cafeteria Lady Elise [5] "What does you sell?", asked Erika [6] "I sell a milk.", answered Cafeteria Lady Elise [7] "I sell a egg.", answered Cafeteria Lady Elise [8] "I sell a hamburger.", answered Cafeteria Lady Elise [9] "I sell a pizza.", answered Cafeteria Lady Elise [10] Erika bought hamburger. -> After that, she decided to buy a hamburger.	[1] "Hello Cafeteria Lady Elise.", greeted Erika [2] "Hello Erika.", greeted Cafeteria Lady Elise [3] "Can I buy food?", asked Erika [4] "You will buy food Erika.", Cafeteria Lady Elise [5] "What does you sell?", asked Erika [6] "I sell a milk.", answered Cafeteria Lady Elise [7] "I sell a egg.", answered Cafeteria Lady Elise [8] "I sell a hamburger.", answered Cafeteria Lady Elise [9] "I sell a pizza.", answered Cafeteria Lady Elise [10] *After that*, she decided to buy a hamburger.

logue. For example, given the story snippet below from *Story 4 (Home, Knowledge, Noun)*, an example interjection "Yehey!" or "Yes!" is added in *line 3* to signify that the child character was happy because they are going shopping. This can be done in a future recommendation by introducing a knowledge representation and algorithm of when to have the characters state supporting interjections between utterances.

```
[1] "What does we do?", asked Dennis
[2] "We will go shopping.", said Mommy Debbie
[3] "Yehey!", exclaimed Dennis
[4] "I will go shopping to Mall.", said Dennis
```

The evaluators also made suggestions to improve the agreement utterance of the characters. The evaluators noted that in order to make character agreements obvious, additional acknowledgement words should be added. For example, instead of Will you play with me? and the answer I will play with you Erika it can be All right, I will play with you Erika or Ok, I will play with you. Another example is with regards to request dialogues, such as Can I buy food? instead of answering You will buy food, a simple Yes or Ok can be done. This has been addressed accordingly in the current system. An example story snippet from a newly created story is given below. *Line 4* shows the modified agreement utterance used by the character.

```
[1] "Hello, Cafeteria Lady Elise,"  greeted Dennis.
[2] "Hello, Dennis,"  greeted Cafeteria Lady Elise.
[3] "May You give vegetable?" asked Dennis.
[4] "Ok ,Dennis,"  agreed Cafeteria Lady Elise.
```

Another suggestion was when another character suggests a solution to the main character, it can be in a form of a question. Instead of ”You will eat in living room”, it can be such as ”Why dont you eat in the living room?”. This has also been implemented in the system. However, the closest the surface realizer can accomplish is the sentence ”Why do you not eat in the living room?”.

Another remark of the evaluators was that politeness should be included in the story when the main character is requesting from another character. This promotes being polite to the target audience, childrens 5 - 8 years of age. For example, when inviting to location, instead of You will come with me, use Can you come with me or Will you come with me?. Also, politeness should also be emphasized when a child character talks to an adult character. For example, the request Can I have some food may also be May I have some food?. This may also improve the authenticity of the dialogues in the story by having children act polite towards his / her elders as well as his / her peers. This issue has been fixed and added in the current system.

The evaluators also suggested to add tag line variations to the stories. For example, the Added tag line can be used for subsequent Answer utterances of the same character in Inquiry dialogues. 3 tag lines ”added”, ”replied”, and ”persuaded” were added in order to cater to this.

6.4.3 Grammar Qualitative Evaluation

Aside from those mentioned in the Quantitative evaluation in improving the articles, prepositions, etc., the evaluators noted that introducing present progressive tense in some sentences may give an improvement to the grammar of the system. An example of this is the Why question. Given the utterance ”Why does you cry?”, it may be improved by making it into present progressive tense such as ”Why are you crying?”. However, this has not currently been implemented because there is an issue with the Surface Realizer being unable to create the correct sentence (”Why are you crying?”. Instead, it creates the sentence ”Why does you crying?”, which seems less appealing compared to the current ”Why does you cry?”.

Table 6.18: Overall Story Evaluation

Research name	Score out of 5
Picture Books 1	4.69 out of 5
Picture Books 2	3.96 out of 5
Story Dialogue Generator	3.4 out of 5

6.5 Comparison with Picture Books 1 and Picture Books 2

In this section, a comparison of the overall story criteria evaluation will be done comparing the works of Picture Books (Hong, Siy, Solis, & Tabirao, 2008), Picture Books 2 (Ang, Antonio, Sanchez, Yu, & Ong, 2010), and this research. The overall story was selected because it is the criteria that deals with the quality of the story. The comparable factors that was presumed to affect the rating of each research will then be discussed.

As seen in **Table 6.18**, Picture Books 1 scored the highest rating of 4.69 out of 5 versus Picture Books 2 with 3.96, and Story Dialogue Generator with 3.4. Picture Books 1 and Story Dialogue Generator used a different rating system from Picture Books 2, in which the rating was only from 1 to 4, while Picture Books 2 had a rating of 1 to 5. To make the rating similar, the ratings of Picture Books 1 and Story Dialogue Generator was adjusted to the rating system of Picture Books 2.

The subsequent subsections will then give an explanation of the possible cause of this rating. The main points of discussion that will be compared are the Story Content and Coherence, and the Characters, Objects, and Backgrounds.

Story Content and Coherence

One factor that may have possible affected the high grade (4.69 out of 5) of Picture Books 1 is because there is already a Predefined Theme that it uses in order to match the Picture created by the user, which is supposed to dictate the story. The predefined theme is composed of the Problem, Rising Action, Solution, and Climax which contains instructions of the events in the story. By having this predefined theme constructed by the authors of the system, it was able to produce a good quality story with good content.

Picture Books 2 scored lower than that of Picture Books 1, with a score of 3.96 out of 5. The possible reason for this is Picture Books 2 does not have a predefined theme given the picture, unlike in Picture Books 1. Instead, Picture Books 2 makes use of a causal chain of actions and events and uses the Pictures dictated by the user as a guide to what happens. Because of this approach to story generation, Picture Books 2 has less control of the Story compared to Picture Books 1. However, actions and events are still connected to each other because of the cause and effect approach. Additionally, an improvement from Picture Books 1 is that Picture Books 2 uses discourse markers, which helped improve the coherence of the story by connecting the sentences to each other.

Finally, the Story Dialogue Generator scored the lowest with a (3.4 out of 5). The possible reason for this is similar to Picture Books 2, it also does not have a complete predefined theme. Using the input of the user, it tries to find the possible path to solve the Character Problem specified, while also using the Background and Lesson inputs to affect the story. One of the major things that affected this rating was possible the absence of no real climax in the Story. Because of the way the Knowledge is represented in the Knowledge Base (CreateProblem, SolveProblem), there are only specific specifications as to the cause of the Inciting Incident and the Falling Action respectively. Additionally, this set-up also causes makes the Inciting Incident and Falling Action not explicitly connected to each other, as the Story Planner only selects the event based on this and not on their actual definition.

The transitions from sentences were also a concern by the evaluators, where some sentences seem unconnected to each other. Additionally, the lack of Discourse Markers also contributes to this problem. A suggestion to improve this was to include more actions / events that may be able to ”fill the gap” between unconnected sentences. An example of this is the story mentioned above where the main character got hurt playing hide and seek, but there was no clear sentence to how she got hurt. Finally, another factor that may have influenced this rating is the transition from Narrative Sentences to Dialogue. Sometimes, it is not clear as to the cause of why the main character chose to engage another character in dialogue. Similar to the problem mentioned, a possible solution to this is to include action / event specifications that help dictate the reason of the character for engaging in dialogue.

Characters, Objects, and Background properties

The next to be discussed will be the interplay of Characters, Objects, and Background properties in the story.

Another factor that may have affected the low grade of the Story Dialogue

Generator was because locations were sometimes confusing. The evaluators noted that sometimes, the locations were a bit confusing. This includes the character going to locations that were not seen as "believable" or unnecessary by the story evaluators. Another is that the Primary Background user input may also confuse the reader, as there are shifts in Primary Backgrounds in the story.

In both Picture Books 1 and Picture Books 2, locations are also specified to guide the story in the formulation of its theme. However, both Picture Books 1 and 2 does not move away from the specified location, and that is a possible reason that the evaluators had no issue with location changes.

Finally, a comparison of the way properties are stored in Picture Books 1 and Picture Books 2 will be made. In Picture Books 1, the introduction of object properties were added near the end of the evaluation and used binary relation to dictate the properties of objects. In Picture Books 2, knowledge about properties about objects were not included, however, the characters themselves has properties or traits (ex. brave). Finally, in the Story Dialogue Generator, properties were stored in a different way. This allows it to be more dynamic in selecting properties compared to Picture Books 1, but the evaluators noted that there was room for improvement when selecting the properties of objects and including them in the story.

6.6 Notable Stories

In this section, notable stories produced by the story generator will be discussed briefly. These stories are the stories that followed the Story Planner algorithm correctly but produced a somewhat incoherent story.

6.6.1 Story 1

In this story, the ATrans selected was the "ask location" ATrans type where knowledge about the location of the needed object is the one needed. This ATrans also triggers a negotiation dialogue. In this story, there are certain story elements that were produced correctly but to unsatisfactory results. First, in *line 29*, because ask location can be done anywhere, it was done in Cafeteria, which may seem wrong because "Food" is already located in Cafeteria. Second, the negotiation dialogue was triggered correctly in *line 30*, and the correct negotiation ("play") action, was also selected properly in *line 30*. However, because "play" can occur in both "Playground" which is in Park, and "Living room" which is in Home, and "play" cannot occur in a secondary background in School, the Story Planner randomly chose "Living room" in *line 30*.

```
[1] The day is sunny.
[2] Michelle is in School.
[3] She is in Classroom.
[4] She has class in Classroom.
[5] She meets Teacher Lady in Classroom.
[6] She talked to Teacher Lady.

[7] "Hello Teacher Lady.",  greeted Michelle
[8] "Hello Michelle.",  greeted Teacher Lady
[9] "What does we do?", asked Michelle
[10] "We will learn.", said Teacher Lady
[11] "What does we learn?", asked Michelle
[12] "We learn about animal.", answered Teacher Lady

[13] Michelle and Teacher Lady learned animal.

[14] "Pig is animal.", answered Teacher Lady
[15] "Wolf is animal.", answered Teacher Lady
[16] "Dog is animal.", answered Teacher Lady

[17] Michelle became hungry.

[18] "I am hungry.", said Michelle

[19] Teacher Lady told Michelle a solution.

[20] "You will eat in Cafeteria.", said Teacher Lady
[21] "I will eat Teacher Lady.",  agreed Michelle

[22] Michelle went to Cafeteria.
[23] Michelle wants to eat.
[24] She needs food to eat.
[25] She meets Dennis, Paul and Cafeteria Lady Elise in Cafeteria.
[26] She talked to Paul.

[27] "Hello Paul.",  greeted Michelle
[28] "Hello Michelle.",  greeted Paul
[29] "Where is food located in?", asked Michelle
[30] "You will play with me first in Living room.", said Paul
[31] "I will play with you Paul.",  agreed Michelle

[32] Michelle and Paul ran to Living room.
[33] Michelle and Paul played.
[34] They rode the bike.

[35] "Food is located in a Cafeteria.", answered Michelle
[36] "Thank you Michelle.",  said Paul
[37] "You are welcome Paul.",  said Michelle
```

```
[38] Michelle wants to eat.
[39] Michelle needs food to eat.
[40] She talked to Dennis.

[41] "Hello Dennis.",  greeted Michelle
[42] "Hello Michelle.",  greeted Dennis
[43] "Can I have food?", asked Michelle
[44] "You will have food Michelle.",  agreed Dennis
[45] "What does you give?", asked Michelle
[46] "I give a fruit.", answered Dennis
[47] "I give a spaghetti.", answered Dennis
[48] "I give a hamburger.", answered Dennis
[49] "I give a pancake.", answered Dennis

[50] Michelle had spaghetti.

[51] "Thank you Dennis.",  said Michelle
[52] "You are welcome Michelle.",  said Dennis

[53] Michelle eats food in Cafeteria.
[54] Michelle felt tired.
[55] She went home.
[56] She learned many things that day.
[57] She learned different animal in Classroom.
[58] She learned different ways to play in Living room.
[59] She learned different food in Cafeteria.
```

6.6.2 Story 2

In this story, what will be discussed is the story content about the agreement and with regards to the Inquiry dialogue that happened. First, in *lines 32 - 37*, it happens that Paul disagreed to all the possible options (”cook”, and ”eat”) suggested by Daddy Carlo. Daddy Carlo is then forced to suggest or persuade Paul to take his previous suggestions (”cook” again). Second, in *lines 47-52*, it seems unrealistic that Paul and Mommy Paula chose to cook Breakfast, Lunch, and Dinner. This problem occurs because the dialogue is trying to find ”conceptuallyRelatedTo” concepts to the concept ”cook”. However, because Breakfast, Lunch, and Dinner has a conceptuallyRelatedTo relation to the concept ”cook”, then they are still selected by the Story Planner.

```
[1] The day is sunny.
[2] Paul is in Park.
[3] He is in Field.
[4] He meets Erika and Michelle in Field.
[5] He talked to to Erika and Michelle.
```

```
[6] "Hello Erika and Michelle.",  greeted Paul
[7] "Hello Paul.",  greeted Erika
[8] "Hello Paul.",  greeted Michelle
[9] "What does we do?", asked Paul
[10] "We will play.", said Michelle
[11] "What does we play?", asked Paul
[12] "We will play baseball.", answered Michelle
[13] "We will play race.", answered Erika
[14] "We will play soccer.", answered Paul

[15] Paul chose to play baseball.

[16] "We will play baseball.", said Paul
[17] "We will play baseball Paul.", Michelle

[18] Paul, Erika and Michelle played baseball.
[19] Paul became hungry.

[20] "Goodbye Erika.",  said Paul
[21] "Goodbye Paul.",  said Erika
[22] "Goodbye Michelle.",  said Paul
[23] "Goodbye Paul.",  said Michelle

[24] Paul walked to Home.
[25] Paul walked to Bathroom.
[26] He meets Mommy Paula and Daddy Carlo in Bathroom.
[27] Daddy Carlo talked to Paul.

[28] "Hello Paul.",  greeted Daddy Carlo
[29] "Hello Daddy Carlo.",  greeted Paul
[30] "I am hungry.", said Paul

[31] Daddy Carlo told Paul a solution.

[32] "You will cook in Kitchen.", said Daddy Carlo
[33] "I do not want to cook Daddy Carlo.", Paul
[34] "You will eat in Kitchen.", said Daddy Carlo
[35] "I do not want to eat Daddy Carlo.", Paul
[36] "You will cook in Kitchen.", said Daddy Carlo
[37] "I will cook Daddy Carlo.", Paul

[38] Paul walked to Kitchen.
[39] Paul does not have someone to cook  with in Kitchen.
[40] He walked to Bathroom.
[41] He talked to Mommy Paula.

[42] "Hello Mommy Paula.",  greeted Paul
[43] "Hello Paul.",  greeted Mommy Paula
[44] "You will come with me to Kitchen.", said Paul
[45] "I will come with you Paul.", Mommy Paula
```

```
[46] Paul and Mommy Paula walked to Kitchen.

[47] "What does we do?", asked Paul
[48] "We will cook.", said Mommy Paula
[49] "What does we cook?", asked Paul
[50] "We will cook dinner.", answered Mommy Paula
[51] "We will cook breakfast.", answered Paul
[52] "We will cook lunch.", answered Mommy Paula

[53] Paul chose to cook breakfast.

[54] "We will cook breakfast.", said Paul
[55] "We will cook breakfast Paul.", Mommy Paula

[56] Paul and Mommy Paula cooked breakfast.
[57] Paul felt tired.
[58] He learned many things that day.
[59] He learned different things to play in Field.
[60] He learned different things to cook in Kitchen.
```

6.6.3 Story 3

In this story, it is notable that it is very short and the characters seem to do just one thing. This story is a sample of a Knowledge story that is very short. Knowledge stories are generally shorter than the other stories (Hunger, and Feel) because knowledge stories are very straight forward in which the main character's goal is to learn something. In Hunger and Feel stories, it is made up of larger events that cause the Hunger and Feel problem. This may be given a solution by including more events in the Knowledge Base that may lengthen the Knowledge stories.

```
[1] The day is sunny.
[2] Dennis is at the Park.
[3] He is at the Playground.
[4] He meets Paul at the Playground.
[5] He talked to Paul.

[6] "Hello, Paul,"  greeted Dennis.
[7] "Hello, Dennis,"  greeted Paul.
[8] "What will we do?" asked Dennis.
[9] "We will see," said Paul.
[10] "What will we see?" asked Dennis.
[11] "We will see playground toy," answered Paul.

[12] Dennis and Paul saw playground toy.
```

```
[13] Dennis sees that slide is playground toy.
[14] Dennis sees that swing is playground toy.
[15] Dennis sees that monkey bar is playground toy.
[16] Dennis felt happy.
[17] He went home.
[18] He learned many things that day.
[19] He learned about different playground toy at the Playground.
```

6.7 Story Breakdown

This section will discuss an example Story and how the Story Planner arrived at that Story. Below is the sample story to be broken down that was created by the Story Generator. The Story created has the User inputs: School for Background, Hunger for CharacterProblem, and Adjective for the Lesson. The whole story will be given first and then followed by the breakdown into the plots that the Story Planner follows.

```
UserInput(School, Hunger, Adjective)

The day is rainy.
Paul is at the school.
He is at the classroom.
He has class at the classroom.
He meets Erika, Michelle and Lady the Teacher at the classroom.
He talked to Erika, Michelle and Teacher Lady.

"Hello, Erika, Michelle and Teacher Lady,"  greeted Paul.
"Hello, Paul,"  greeted Erika.
"Hello, Paul,"  greeted Michelle.
"Hello, Paul,"  greeted Teacher Lady.
"What will we do?" asked Paul.
"We will learn," said Teacher Lady.
"What will we learn?" asked Paul.
"We will learn about job," answered Teacher Lady.

Paul, Erika, Michelle and Teacher Lady learned job.

"Doctor is a job," answered Erika.
"Doctors are helpful,"  added Erika.
"Nurse is a job," answered Michelle.
"Nurses are good," said Erika.
```

"Police officer is a job," answered Michelle.
"Police officer are helpful," said Erika.
"Dentist is a job," answered Teacher Lady.
"Dentists are good," said Erika.

Paul became hungry.

"I am hungry," said Paul.

Teacher Lady told Paul a solution.

"Why do you not eat at the cafeteria?" asked Teacher Lady.
"Ok, Teacher Lady," agreed Paul.
"Thank you, Teacher Lady," said Paul.
"You are welcome Paul," replied Teacher Lady.
"Goodbye, Erika and Michelle," said Paul.
"Goodbye, Paul," replied Erika.
"Goodbye, Paul," replied Michelle.

Paul ran to the cafeteria.
Paul wants to eat.
He needs food to eat.
He meets Dennis and Elise the Cafeteria Lady at the cafeteria.
He talked to Dennis.

"Hello, Dennis," greeted Paul.
"Hello, Paul," greeted Dennis.
"Can You give me fruit?" asked Paul.
"Ok, Paul," agreed Dennis.
"You will play with me first at the playground," added Dennis.
"Ok, Dennis," agreed Paul.

Paul and Dennis ran to the playground.
They played.
They smoothly rode the long bike.
They loudly used the ugly slide.
Dennis gave Paul a fruit.

"Thank you, Paul," said Dennis.
"You are welcome Dennis," replied Paul.

Paul returned to the school.

Table 6.19: Exposition

Story Text	Planner rules	Relations used
The day is rainy.	findEvent(createCharProblem(	createCharProblem(discuss,Hunger)
Paul is at the school.	<Action> , Hunger)	==> locationOf(discuss,classroom)
He is at the classroom.	==> discuss, read, see	createCharProblem(read,Hunger)
	==> random(discuss, read)	==> locationOf(read,classroom)
	==> discuss	createCharProblem(see,Hunger) ==>
		locationOf(see,bookstore)
	executeEvent(discuss)	hasSecondaryBackground(School ,
	executeFirstEvent(learn)	classroom)
	executeFirstEvent(have class)	
		hasFirstSubEvent(discuss,learn)
	executeEvent(have class)	hasFirstSubEvent(learn,have class)
	==> Set exposition	
	Assign character locations	conceptuallyRelatedTo(weather,
	currentlyInLocation(Paul,classroom)	rainy)
	currentlyInLocation(Erika,classroom)	hasSecondaryBackground(School,
	currentlyInLocation(Michelle,classroom)	Classroom)
	currentlyInLocation(Lady,classroom)	

```
Paul returned to the cafeteria.
He eats fruit at the cafeteria.
He felt happy.
He went home.
He learned many things that day.
He learned about different job at the classroom.
He learned about different ways to play at the playground.
```

6.7.1 Exposition and Inciting Incident

Table 6.19 and **Table 6.20** gives the breakdown of the Exposition and Inciting Incident respectively. This section will provide a discussion of how the Story Planner created the Expostion and Inciting Incident.

First, after the user has given the input to the Story Planner (School, Hunger, Noun), the Story Planner tries to find events that can CREATE the Character Problem (Hunger) that the user has input. In this example, a sample subset of the events that can CreateHunger is the actions / events Discuss, Read, and See. The Story Planner then selects from this subset. First, it already eliminates the See action because it has a different location from that of the Background (School) that was given as input by the user. However, Discuss and Read are both retained because they occur in Classroom and Classroom or Library respectively, which can be found in the School. The Story Planner then randomly selects from the possible choices, Discuss and Read, and in this case, selects the Discuss event.

Table 6.20: Inciting Incident

Story Text	Planner rules	Relations used

He has class at the classroom.
He meets Erika, Michelle and Lady the Teacher at the classroom.
He talked to Erika, Michelle and Teacher Lady.
 	
"Hello, Erika, Michelle and Teacher Lady," greeted Paul.
"Hello, Paul," greeted Erika.
"Hello, Paul," greeted Michelle.
"Hello, Paul," greeted Teacher Lady.
"What will we do?" asked Paul.
"We will learn," said Teacher Lady.
"What will we learn?" asked Paul.
"We will learn about job," answered Teacher Lady.
 	
Paul, Erika, Michelle and Teacher Lady learned job.
 	
"Doctor is a job," answered Erika.
"Doctors are helpful," added Erika.
"Nurse is a job," answered Michelle.
"Nurses are good," said Erika.
"Police officer is a job," answered Michelle.
"Police officer are helpful," said Erika.
"Dentist is a job," answered Teacher Lady.
"Dentists are good," said Erika.
 	
Paul became hungry.
	executeEvent(learn)
	executeDialogue(InformationSeeking)
	executeDialogue(InformationSeeking)
	==> Job
	
	executeEvent(discuss)
	executeDialogue(Inquiry)
	==> findTopicRelation(Job)
	==> isA
	==> Doctor, Nurse, Police Officer, Dentist
	==> propertyOf
		hasRequirements(learn,askDo)
		triggerDialogue(learn,Information-Seeking(IS;What;NounInstance))
		conceptuallyRelatedTo(learn, job)
		
		triggerDialogue(discuss,Inquiry)
		isMTransType(discuss,CharHear)
		isGeneralNoun(job, isA)
		isA(Doctor, Job)
		isA(Nurse, Job)
		isA(Police Officer, Job)
		isA(Dentist, Job)
		propertyOf(Dentist,QualityLiving)
		Modifier(helpful,QualityLiving)
		propertyOf(Nurse,QualityLiving)
		Modifier(good,QualityLiving)

Having selected the Inciting Incident, the event Discuss will then be executed in the Story Executor.

executeEvent(discuss)

The Story Executor then tries to execute the Discuss event. However, it finds that the Discuss event has a first sub event learn. The Story Executor then tries to execute the event learn. It then finds that the action learn also has a first sub event of have class. Therefore, the Story Executor tries to execute Have Class first. Finally, the Story Executor finds that the event have class does not have a first sub event and can already be executed. Since the event have class is also the very first event that was executed, the Story Executor sets the Exposition of the Story based on where the very first event have class occurred (which is in classroom).

In setting the exposition of the Story, it sets the setting of the day and the location of the characters in the Primary background (School) where the first event occurs. In this case, Paul, Erika, and Michelle was randomly assigned to the Classroom secondary background which is located in the School.

After executing the event have class, it then goes back to execute the event learn.

The event learn has a requirement, askDo, which requires to call an Information Seeking dialogue. After executing the askDo requirement, the Story Executor then finds that the learn event triggers an InformationSeeking dialogue. This specific InformationSeeking dialogue asks about a specific Topic to which the action learn can be applied to. In this case, it uses the conceptuallyRelatedTo relation and consults the KB for related concepts. It then finds 3 possible concepts, namely, Job, Animals, and Food. It then randomly selects Job.

Finally, after executing the learn event, the Story Executor can now execute the Discuss event. The Story Executor finds that the Discuss event triggers an Inquiry dialogue.

executeDialogue(Inquiry)

Using the Topic provided by the previous event, the Discuss event executes its Inquiry dialogue. First, it consults the KB for its MTrans type, which is found to be CharHear. This helps form the format of the Inquiry dialogue. Second, it finds the applicable relations that the Inquiry dialogue can discuss about the

Topic, in this case Job. It finds that the relation isA is applicable, while the relations made Of and part Of is not. After knowing that the relation to be used is isA, it then finds related concepts to Job using the relation isA. It then finds the concepts Doctor, Nurse, Dentist, and Police Officer. Finally, because the lesson input of the user is Adjective, it then tries to find properties of the selected answers Doctor, Nurse, Dentist, and Police Officer. It finds that these concepts can have the property f QualityLiving. It then finds Modifiers that can be applied to the QualityLiving property. It finds that the modifiers, helpful, and good can be applied to this.

Finally, because the Discuss event is already done, the Story Executor then creates the Hunger problem and finishes of the Inciting Incident part of the planning.

6.7.2 Climax and Falling Action

Table 6.21 and **Table 6.22** gives the breakdown of the Climax and Falling Action respectively. This section will provide a discussion of how the Story Planner created the Climax and Falling Action.

After finding an event that can create the problem Hunger, the Story Planner needs to find an event that SOLVES the problem Hunger which is used to form the Falling Action and Climax parts of the story. In this case, it finds 3 possible events, Suggest Solution, Eat, and Cook that can SolvesHunger. However, it eliminates the Cook event because it is to be executed in Kitchen which is not located in the current Primary Background School. It then randomly selects from the remaining choices Suggest Solution and Eat and randomly selects Suggest Solution.

executeEvent(Suggest Solution)

The Story Executor then executes the "Suggest Solution" event. It finds that it triggers a Persuasion;Suggest Solution Dialogue using the *triggersDialogue* relation. It then executes the Persuasion dialogue with the path Suggest Solution. The Suggest Solution path then finds events that can solve the character problem Hunger excluding itself in order to suggest to the main character. It finds the events "eat" and "cook". It then randomly selects the event "eat" to be the suggested event to the main character.

Table 6.21: Climax

Story Text	Planner rules	Relations used

”I am hungry,” said Paul.
 	
Teacher Lady told Paul a solution.
 	
”Why do you not eat at the cafeteria?” asked Teacher Lady.
”Ok, Teacher Lady,” agreed Paul.
”Thank you, Teacher Lady,” said Paul.
”You are welcome Paul,” replied Teacher Lady.
”Goodbye, Erika and Michelle,” said Paul.
”Goodbye, Paul,” replied Erika.
”Goodbye, Paul,” replied Michelle.
 	
Paul ran to the cafeteria.
Paul wants to eat.
He needs food to eat.
He meets Dennis and Elise the Cafeteria Lady at the cafeteria.
He talked to Dennis.
 	
”Hello, Dennis,” greeted Paul.
”Hello, Paul,” greeted Dennis.
”Can You give me fruit?” asked Paul.
”Ok, Paul,” agreed Dennis.
”You will play with me first at the playground,” added Dennis.
”Ok, Dennis,” agreed Paul.
 	
Paul and Dennis ran to the playground.
They played.
They smoothly rode the long bike.
They loudly used the ugly slide.
Dennis gave Paul a fruit.
 	
”Thank you, Paul,” said Dennis.
”You are welcome Dennis,” replied Paul.
	findEvent(solveCharProblem(<Action>, Hunger)
	==> Suggest Solution, eat, cook
	==> random(Suggest Solution, eat)
	==> Suggest Solution
	
	executeEvent(SuggestSolution)
	executeDialogue (Persuasion ; SuggestSolution)
	==> findEvent(solveCharProblem(<Action> , Hunger)
	==> eat, cook
	==> random(eat, cook)
	==> eat
	
	executeEvent(eat)
	==> hasRequirements(NeedObject)
	==> findATransAction
	==> ask, ask choice, buy
	==> random(ask, ask choice, buy)
	==> ask
	executeATransAction(ask)
	executeDialogue(Negotiation)
	==> findEvent(isNegotiationEvent(<Action> , Child))
	==> play
		solveCharProblem(Suggest Solution,Hunger)
		==> locationOf(Suggest Solution, anywhere)
		solveCharProblem(eat, Hunger)
		==> locationOf(eat, cafeteria)
		solveCharProblem(cook, Hunger)
		==> locationOf(cook, kitchen)
		random(Suggest Solution, eat)
		==> Suggest Solution
		executeEvent(Suggest Solution)
		triggersDialogue(SuggestSolution , Persuasion ; SuggestSolution)
		solveCharProblem(eat, Hunger)
		solveCharProblem(cook, Hunger)
		random(eat, cook)
		==> eat
		
		executeEvent(eat) hasRequirements(eat,NeedObject(food))
		
		locationOf(ask, anywhere)
		locationOf(ask choice, anywhere)
		locationOf(buy, cafeteria)
		isATransType(ask, specificInstance)
		==> isA(food,fruit)
		canTriggerNegotiation(ask,true)
		isNegotiationEvent(play,child)
		
		howTo(play,ride)
		howTo(play,use)
		capableOf(ride,bike)
		capableOf(use,slide)
		propertyOf(ride,adverb)
		Modifier(smooth,adverb)
		propertyOf(use,adverb)
		Modifier(loud,adverb)
		propertyOf(bike,size)
		Modifier(long,size)
		propertyOf(slide,appearanceObject)
		Modifier(ugly,appearanceObject)

Table 6.22: Falling Action

Story Text	Planner rules	Relations used
Paul returned to the school. Paul returned to the cafeteria. He eats fruit at the cafeteria. He felt happy.	executeEvent(eat)	

executeEvent(eat)

The Story Planner will then execute the suggested event "eat". First, it finds that it has a requirement of NeedObject(Food). Because of this, it then searches for an ATrans action that can be used in order to gain the needed object "Food". It finds the events "ask", "ask choice", and "buy". As the events can be all executed in the current Primary Background (School), the Story Planner randomly selects from the events. It randomly selects the ATrans action "ask".

executeATransAction(ask)

The StoryPlanner finds that the "ask" action has an ATransType of specificInstance. Because of this, it first finds an instance of the needed object "Food" using the *isA* relation, finding "fruit" to be a related concept. It then finds that the action "ask" also triggers a Negotiation dialogue. Because of this, it executes the Negotiation dialogue.

executeDialogue(Negotiation)

First, the Story Planner finds an action / event that will be used as the proposition with the negotiator being a child using the *isNegotiationEvent* relation. It then finds the action / event "play" to be a suitable action / event. It then executes the action / event using a series of relations. First, it uses the *howTo* relation to finds concepts related to the action "play" and finds the concepts "ride" and "use". Then it finds objects that the *howTo* concepts can be applied to using the *capableOf* relation and finds the objects "bike" and "slide" for "ride" and "use" respectively. Finally, since the lesson selected by the user is Adjective, additional queries are made using the *propertyOf* relation for both the *howTo* and *capableOf* concepts selected.

After executing the Dialogue, the negotiator then gives the NeededObject to the main character and the main character is then free to execute "eat".

Table 6.23: Conclusion

Story Text	Planner rules	Relations used

He went home.
He learned many things that day.
He learned about different job at the classroom.
He learned about different ways to play at the playground.
	triggersDialogue(Inquiry)
	==> Job
	triggersDialogue(Negotiation)
	==> play	

6.7.3 Denouenment / Conclusion

Finally, the Conclusion is formed by the Story Planner. **Table 6.23** gives the breakdown of the Conclusion.

During the Story Planning phase, it stores the topics that were done in depth, specifically when it triggered the Inquiry and Negotiation dialogues. Therefore, it knows that the main character learned about "Jobs" using the Inquiry dialogue and the different ways "play" using the Negotiation dialogue.

7 Conclusion and Future Recommendations

This section discusses and explains the summary of accomplished requirements and also discusses the possible recommendations for the improvement of the application.

7.1 Conclusion

The main purpose of this research is to design a story generation system that will be able to generate Stories with Dialogues. In order to create this system there are two (2) main modules that was needed: (1) The Story Planner which contains the algorithm in order to create stories and (2) The Knowledge Base used by the Story Planner in order to get the knowledge required to create stories with dialogues.

Based on the results of the Overall Story evaluation, the story generator was able to produce acceptable stories with dialogues with an overall rating of 2.68 over 4. Dialogues are linked to the actions / events in the Knowledge Base using a specific field in the knowledge base. This approach allows for tailoring events in the Knowledge Base to which events actually trigger a specific kind of Dialogue. Based on the results, this approach was able to produce acceptable stories with dialogues. Also, the usage of the Character Problems and Character Actions also helped in creating acceptable stories. The stories were also found to be appropriate to the target audience, which is children ages 5 - 8.

Based on the results of the Dialogue evaluation, using Utterances for defining the Dialogue Types proved useful in defining the structure of each dialogue depending on its purpose as it was able to help the story generator to generate acceptable dialogues in the story with an overall rating of 2.80 over 4. Even just using adapting only 5 basic utterances from the DAMSL scheme was able to form the dialogues in the story.

Finally, the inclusion of the Lesson in the dialogues also proved to be quite effective as the evaluators noted that the Stories were able to somewhat give the Lesson in the stories it creates.

7.2 Future Recommendations

Improve Story Planning

One of the remarks of the evaluators was that the Overall Story may still be improved. One of the notable elements that the stories lacked as said by the evaluators was the inclusion of a clear climax. The current design of the Knowledge Base focused more on being able to specify events that can be used as the Inciting Incident or the Falling Action. The events that happen between the said events is implicitly the climax of the story. This can be improved in future works where the Knowledge Base and Story Planner will be designed to also handle the creation of Climax in the story.

Improve Overall Story Coherency

Another future work that can be done is to improve the coherency of the stories, as it was also another element that the evaluators noted that may be improved. This can be done by improving the relation of the events in the story to one another. There are 2 ways that this can be done: (1) by designing / creating an algorithm that will create stronger connections between events and (2) by introducing the use of Discourse Markers in the story.

Improve Narrative to Dialogue transition

Additionally, the evaluators also noted that transition from narrative to dialogue were sometimes unclear as to the motivation of the characters to engage in dialogue. In order to improve this, a possible future work is to define an algorithm that will help the Story Planner understand more on when to trigger certain types of dialogues. In doing this, it may also be able to provide the necessary transitions in the form of dialogue or simple sentences, that will give more detail as to the purpose of the start of the dialogue.

Dialogue participant selection

Another improvement that can be made is with regards to the dialogue participants selection. Similar to how dialogues are mapped to events, the dialogue participants (Dialogue Speaker and Dialogue Hearer) are also mapped into the events in the knowledge base. This design makes the inputting of dialogue participants in actions / events to be quite static and time-consuming as they are manually added. An improvement might be to design an algorithm for both the Knowledge Base and Story Planner that allows the Story Planner to understand who the dialogue participants will be in an event.

Addition of themes to the story

Another suggestion of the evaluators was the inclusion of themes to the story. The addition of themes might help improve the completeness of the plot of the story. For example, instead of having a story just discussing about "Food", there can be a theme set such as discussing about "Healthy Foods". This in turn also promotes healthy foods to the target audience which is children ages 5 - 8. The addition of themes may also improve the story in the conclusion plot where insight / knowledge / moral of the story in the conclusion is presented depending on the story events (ex. eating junk foods, having a toothache, and eating "Healthy Foods" instead of Junk foods). Additionally, a rather trivial addition may be to automatically generate the *Title* of a story by using the Theme of the story.

Using Personalities to affect Dialogues

One of the future works that can be done to the system is the introduction of Personalities to the characters in the story and using the Character Personalities to influence the utterances produced by the characters.

A good start to do this future work is to consult the works of Oberlander and Gill (2004), and Goldberg (1993). The Critical Agent Dialogue (CRAG) project (Oberlander & Gill, 2004) studied how personality affects language production, using the Big Five personality traits. The Big Five personality traits are the dimensions of personality that have been developed by lexical analysis of words and how they relate to personality (Goldberg, 1993).

Inclusion of more types of dialogues

Currently, the system only includes the five of the six types of dialogues mentioned by Walton (1992) and Walton and Macagno (2007). The Eristic dilaogue has been left out because the Eristic dialogue is more useful for affecting character disposition towards one another, which falls under the field of Affective text generation, similar to that of the inclusion of Personalities of characters within the stories.

Aside from including the Eristic dialogue, a good future work would also be to find more types of dialogues and the purposes of each dialogue. Each dialogue will then be implemented into the story generator to allow it to produce more dynamic and entertaining dialogue between characters in the story.

Inclusion of more utterances

Similar to the inclusion of more types of dialogues, another possible future work is to include more types of utterances defined by the the DAMSL annotation scheme (Core & Allen, 1997). Currently in this system, only the basic utterances were used as they were already sufficient enough to form the types of dialogues currently included in the story. However, through the introduction of new types of dialogues in the story, more utterances might need to be used in order to properly create the new types of dialogue introduced in the story.

Usage of a Dialogue History

Through the introduction of new utterances, a Dialogue History may need to be designed for some of the more complicated Utterances. An example is the *Understanding* utterance stated by the DAMSL annotation scheme where a character signals to the other character if he / she understood his / her proposal. A Dialogue History might be useful in order to allow the Story Planner to traverse the utterances spoken and do the appropriate action.

Usage of more types of Character Problems

Currently there are only three (3) types of Character Problems that serve as the main guide for the Story Planner to generate the stories. This may limit the possible types of stories that the Knowledge Base will be able to create. In order to allow the Knowledge Base to be able to create more varieties of stories, additional character problems may be added.

Research as an Educational Tool

Because the domain of the stories revolve around introducing to the reader the concepts for vocabulary acquisition in the English Language, specifically the parts of speech in a sentence which are "Nouns", "Verbs", and "Adjectives", the research may possibly have a future in being used as an educational tool. When a child identifies with a specific character in the story, he/she may also mimic that characters behavior as well as language. The stories generated may also enrich the vocabulary of the child. Therefore, a possible future work for the research is to modify the system a bit in order to be able to be used as an Educational Tool for children.

Better Grammar

Finally, the last possible future work is improving the grammar of the Story Generator. The evaluators noted that the stories may be improved by improving the grammar of the story. This can be done in two ways: (1) Making more complex sentences that has quality comparable to author written stories and (2) improving word contexts, pronouns, articles, and the use of discourse markers throughout the story.

For the first improvement, the current Story Generator is only able to produce simple sentences. A good future research would be to focus more on creating complex sentences (similar to the research of Callaway and Lester (2002)) with good quality. This includes both the narrative sentences and dialogues.

For the second improvement, it is with regards to the improvement of the word contexts, pronouns, articles, and the use of discourse markers throughout the story. The evaluators has noted that there were some wrong grammar with regards to those elements. The main reason that there are some errors with regards to the correct usage of articles and prepositions is because the articles and prepositions are added on Sentence Specifications depending on where they have been called in the Story Planner. In order to improve this, further study can be made on how to properly select the needed articles or prepositions. A possible solution to this is to use the relations used in the Knowledge Base to specify the correct article / preposition. For example, the relation "canDo" can be used to define what instrument is used to do an action. An example is "knife" canDo "chop", can then be used to "Paul chopped vegetable with a knife". However, this might need further research on what relations define what articles and prepositions. Additionally, other relations might need to be added in order to define other prepositions / articles.

References

Ang, K., Antonio, J., Sanchez, D., Yu, S., & Ong, E. (2010). Generating stories for a multi-scene input picture. In *Proceedings of the 7th national natural language processing research symposium, 21-26, november 19-20 2010, de la salle university, manila.*

Callaway, C. B., & Lester, J. C. (2002, August). Narrative prose generation. *Artif. Intell.*, *139*, 213–252. Available from `http://portal.acm.org/citation.cfm?id=635012.635015`

Core, M., & Allen, J. (1997). Coding dialogues with the damsl annotation scheme. In *In proceedings of the aaai fall 1997 symposium on communicative action in humans and machines. american association for artificial intelligence, 1997.*

Dils, T. (1998). *You can write children's books.* Cincinnati, Ohio: Writer's Digest Book.

Goldberg, L. (1993, January). The structure of phenotypic personality traits. In *The american psychologist* (Vol. 48, p. 26-34).

Hong, A., Siy, J., Solis, C., & Tabirao, E. (2008). *Picture books: An automated story generator.* (Manila, Philippines: Undergraduate Thesis)

Liu, H., & Singh, P. (2004, October). Conceptnet a practical commonsense reasoning tool-kit. *BT Technology Journal*, *22*, 211–226. Available from `http://portal.acm.org/citation.cfm?id=1031314.1031373`

Meehan, J. R. (1977). Tale-spin, an interactive program that writes stories. In *Proceedings of the 5th international joint conference on artificial intelligence - volume 1* (pp. 91–98). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Available from `http://portal.acm.org/citation.cfm?id=1624435.1624452`

Oberlander, J., & Gill, A. J. (2004). Individual differences and implicit language: personality, parts-of-speech, and pervasiveness. In *Proceedings of the 26th annual conference of the cognitive science society* (p. 1035-1040). Chicago, IL.

Ramet, A. (2001). *Creative writing : use your imagination, develop your writing skills and get published.* Oxford, England: How to Books.

Schank, R. C. (1975). The primitive acts of conceptual dependency. In *Proceedings of the 1975 workshop on theoretical issues in natural language processing* (pp. 34–37). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dx.doi.org/10.3115/980190.980205`

Walton, D. (1992). Types of dialogue, dialectal shifts and fallacies*. In *Argumentation illuminated* (Frans H. van Eemeren et al ed., p. 133-147). Amsterdam, SICSAT.

Walton, D., & Macagno, F. (2007). Types of dialogue, dialectical relevance, and textual congruity. In *Anthropology philosophy: International multidisciplinary journal* (8th ed., p. 101-119).